# Budgeting for Synthesis
User Guide

Version X-2005.09, September 2005

Comments?
Send comments on the documentation by going to http://solvnet.synopsys.com, then clicking “Enter a Call to the Support Center.”

SYNOPSYS®

Printed in the U.S.A.

Document Order Number: 37656-000 ZA
Budgeting for Synthesis User Guide, version X-2005.09

# Contents

# Preface

This preface includes the following sections:

## What's New in This Release

Information about known problems and limitations, as well as about resolved Synopsys Technical Action Requests (STARs), is available in the *Design Compiler Release Notes* in SolvNet.

To see the *Design Compiler Release Notes*,

1. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNet.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)
3. Click Release Notes in the Main Navigation section (on the left), click Design Compiler, then click the release you want in the list that appears at the bottom.

## About This User Guide

The *Budgeting for Synthesis User Guide* describes the methodology for performing design budgeting (constraint allocation) using the synthesis design budgeter.

### Audience

This user guide is for engineers who use Design Compiler for optimizing designs. You need to be familiar with

- Logic design and timing principles
- Synopsys synthesis tools

- The UNIX operating system

If you are using a platform other than a Sun workstation, you might see minor differences in the way you interact with your operating system.

## Related Publications

For additional information about the Design Compiler budgeter, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys Electronic Software Transfer (EST) system
- Documentation on the Web, which is available through SolvNet at http://solvnet.synopsys.com
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at http://mediadocs.synopsys.com

These Synopsys documents provide additional information:

- *Automated Chip Synthesis User Guide*
- *Design Compiler Command-Line Interface Guide*
- *Design Compiler Reference Manual: Constraints and Timing*
- *Design Compiler Reference Manual: Optimization and Timing Analysis*
- *Design Compiler User Guide*
- *Installation Guide*

## Conventions

The following conventions are used in Synopsys documentation.

| Convention | Description |
|---|---|
| `Courier` | Indicates command syntax. |
| *`Courier italic`* | Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.) |
| **`Courier bold`** | Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.) |
| [ ] | Denotes optional parameters, such as *`pin1`* `[`*`pin2 ... pinN`*`]` |
| \| | Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.) |
| _ | Connects terms that are read as a single term by the system, such as `set_annotated_delay` |
| Control-c | Indicates a keyboard combination, such as holding down the Control key and pressing c. |
| \ | Indicates a continuation of a command line. |
| / | Indicates levels of directory structure. |
| Edit > Copy | Indicates a path to a menu command, such as opening the Edit menu and choosing Copy. |

## Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

### Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call With the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at http://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click Help in the column on the right side of the SolvNet Web page.

## Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), then clicking “Enter a Call With the Support Center.”
- Send an e-mail message to support_center@synopsys.com.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

# 1

# Introduction to Budgeting for Synthesis

In general, the best strategy for compiling designs is to use a top-down compile approach because Design Compiler automatically optimizes slack allocation for the entire design. However, some designs cannot use this because of capacity limitations. For best results in these cases, you have to create budgets for your subblocks and compile them separately. Synthesis budgeting automates the process of allocating top-level timing and environmental constraints to lower-level design blocks.

Synthesis budgeting is introduced in these sections:

- Benefits of Budgeting
- Designs Best Suited for Budgeting
- General Budgeter Capabilities
- Starting and Exiting the Budgeter

## Benefits of Budgeting

The following three methods are used to develop lower-level constraints from top-level design specifications:

- The manual method

  Generates detailed block-level synthesis constraints manually

- The characterization method

  Uses the `characterize` command to calculate the actual attributes and constraints imposed on a cell instance by its surroundings and places those constraints on the cell instance

- The budgeting method

  Uses `dc_allocate_budgets` to automatically generate lower-level constraints for all blocks

These methods are described in Table 1-1.

*Table 1-1 Description of Budgeting Methods*

| Budgeting Method | Description and Limitations |
|---|---|
| Manual method | You generate detailed block-level synthesis constraints manually according to the interblock requirements approved by designers. Generating constraints manually is laborious and can lead to overconstraining, underconstraining, or missing constraints entirely. |
| Characterization method | This method requires a design that is mapped to gates. RTL designs cannot use this method. The characterization method requires that you characterize a block, compile it, characterize another block, compile it, and so forth.<br><br>The `characterize` command extracts the timing context and assigns the total slack of a path to the block being characterized; it does not perform slack allocation. Assigning the total slack of a path to the block being characterized can lead to the block being overconstrained and, consequently, to larger circuits and longer runtimes.<br><br>The result obtained from the serial characterize compile strategy is determined by the order in which the blocks are characterized. This order can be hard to determine because different path constraints prefer different orders. |
| Budgeting method | This method can budget RTL designs, mapped designs, and designs that contains a combination of both RTL and mapped blocks.<br><br>The budgeting method uses the `dc_allocate_budgets` command to automatically allocate top-level constraints to all lower-level blocks so the top-level timing goals are met if each block meets its timing goals after synthesis.<br><br>During budget allocation, the budgeter analyzes the timing of the design and allocates delay, design rule, and environment budgets for cells or pins. You can allocate budgets automatically, relying on the tool to perform allocation, or you can direct the allocation. |

Budgeting solves the limitations of using the characterize method by

- Providing better QoR

  Using a methodology that employs automated budgeting enables you to obtain better QoR in less time. Compared to a methodology using the serial characterize compile strategy, budgeting typically produces better results and allows you to compile designs in parallel.

- Providing greater ease of use

  Budgeting provides a streamlined methodology for generating block constraints from chip-level timing objectives.

- Enabling faster turnaround time

  Turnaround time is the elapsed time between starting the synthesis step and obtaining a netlist that meets requirements. The budgeting methodology enables you to perform compile runs in parallel, significantly reducing turnaround time for large designs, as shown in Figure 1-1 and Figure 1-2.

*Figure 1-1 Design for Comparing Characterize and Budgeting*

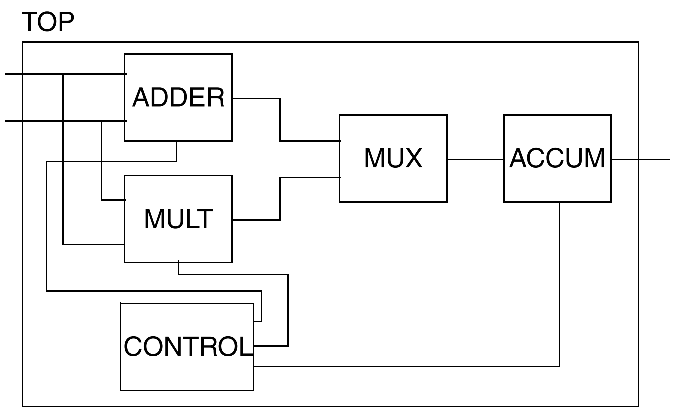
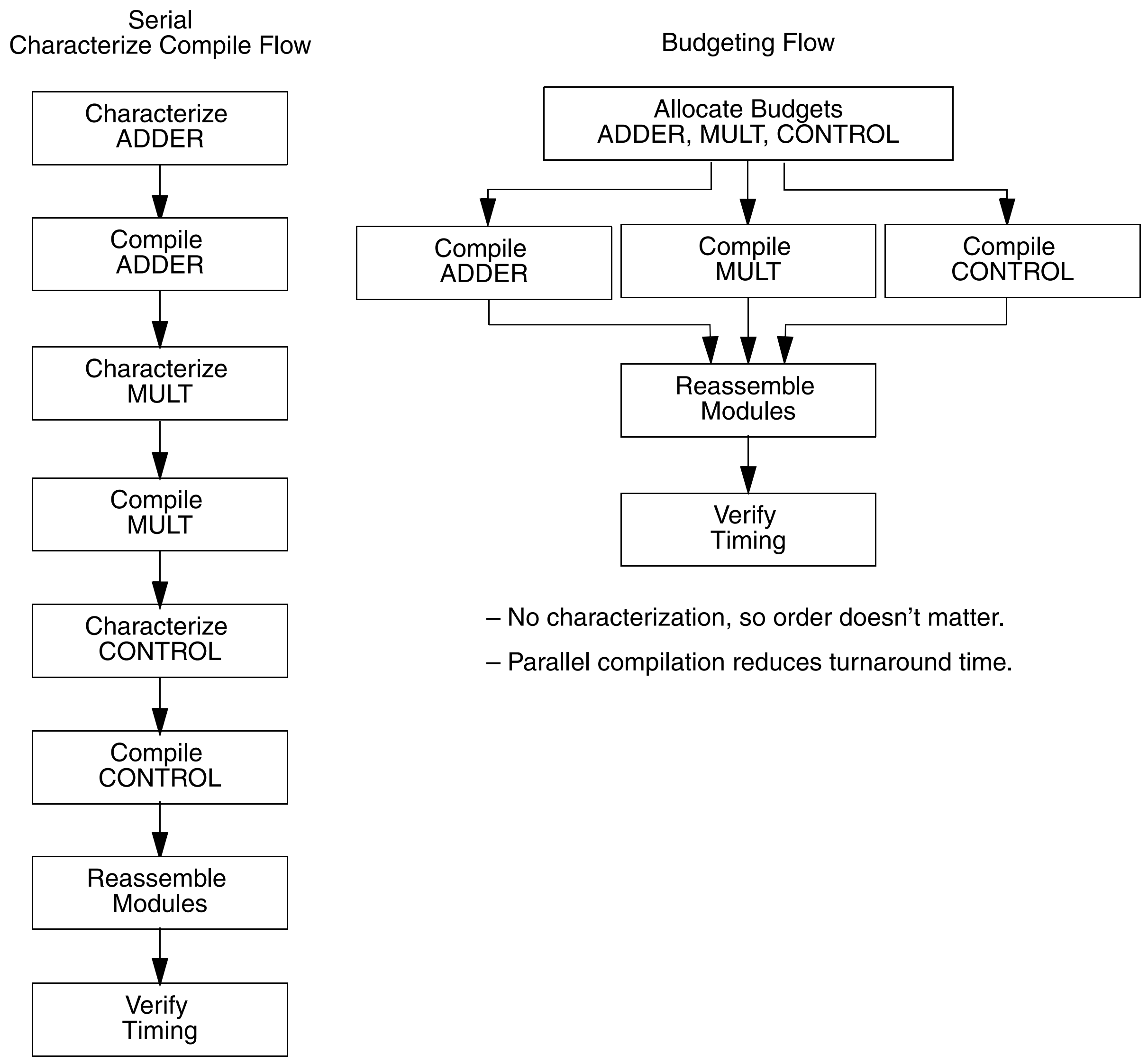


*Figure 1-2 Comparing Turnaround Times for Characterize and Budgeting*



– QoR depends on the order in which blocks are characterized.

– Each characterize and compile depends on the previous characterize and compile.

– Serial nature leads to long turnaround times.

– No characterization, so order doesn’t matter.

– Parallel compilation reduces turnaround time.

## Designs Best Suited for Budgeting

Budgeting is most effective for the following types of designs:

- Hierarchical designs

  Budgeting is effective only when the design contains sufficient hierarchy so that reasonable budgets can be created.

- Large designs

  The budgeter works at the chip level with designs that are more than approximately 100K cell instances. If the design is small enough to compile top down, continue to use that approach because the budgeter does not provide better QoR than a top-down approach for small designs.

- Designs with subblocks that are not fully registered (designs with “snake” paths)
- Optimized blocks that are not meeting timing

In addition to the overall design requirements listed above, observe the following subblock guidelines:

- Balance the hierarchy

  When you choose blocks to budget, you need to decide the size of the block you intend to compile as a separate unit (for example, 100K, 150K, or 50K). For example, if the top level contains a 100K block, a 20K block, and five 3K blocks, you might want to budget blocks within the 100K block that do not exceed 50K gates.

- Use suitable block sizes

You need a block size suitable for hierarchical compilation. Generally, this size is from 30K to 80K gates.

- Keep related logic together

  Partition your design to keep related logic for each functional unit in the same block.

## General Budgeter Capabilities

In budgeting, each timing path is allocated a percentage of the clock period based on a budgeting algorithm.

The budgeter has these general capabilities:

- Considers physical information (back-annotated parasitics and back-annotated wire delays)

  See Chapter 4, "Budgeting With Back-Annotated Information."

- Provides budget analysis and reporting tools

  See Chapter 5, "Analyzing Your Budget and Solving Problems."

- Handles timing exceptions

  See "Supported Constraints" on page 1-9.

- Supports timing models

  See "How Timing Models Are Budgeted" on page 5-9.

- Supports user-directed budget allocation

  See Chapter 3, "Interactive Budgeting."

- Supports multiple instantiations of a design
- Generates constraints for Design Compiler in dc_shell, dc_shell-t, and dc_shell-xg-t formats

  See "dc_allocate_budgets" on page 7-5.
- Performs target library analysis. This results in realistic initial delays to GTECH cells: an unmapped cell delay is comparable to its mapped equivalent cell.

## Starting and Exiting the Budgeter

You need not explicitly start or exit the budgeter. Issuing the `dc_allocate_budgets` command within the Design Compiler shell runs the budgeter.

## Supported Constraints

The budgeter generates block-level synthesis constraints so that the top-level timing goals are met if each block meets its timing goals after synthesis. The constraints generated for each block include

- Timing constraints, including arrival times on block inputs and required times on block outputs and timing exceptions
- Environment constraints, including capacitance on block inputs and block outputs, target number of fanouts, and driving cell information for block inputs
- Design rules, including `max_capacitance`, `max_transition`, and `max_fanout` on block inputs and block outputs

- Synthesis directives, applied at the chip level and inherited by affected individual blocks (examples include `dont_touch` attributes, equal-opposite attributes, and so forth)

Table 1-2 lists the types of constraints generated by the budgeter. In addition, the budgeter generates constraints from the obsolete but supported commands `set_clock_skew` and `set_wire_load`.

*Table 1-2 Constraints Generated by the Budgeter*

| **Type of information** | **Constraint commands** |
|---|---|
| Operating conditions | `set_operating_conditions` |
| Wire load models | `set_wire_load_min_block_size`<br>`set_wire_load_mode`<br>`set_wire_load_model`<br>`set_wire_load_selection_group` |
| Synthesis directives | `set_dont_touch`<br>`set_dont_touch_network`<br>`set_dont_use`<br>`set_fix_hold`<br>`set_fix_multiple_port_nets`<br>`set_ideal_network`<br>`set_ideal_net`<br>`set_max_area` |
| System interface | `set_drive`<br>`set_driving_cell`<br>`set_fanout_load`<br>`set_input_parasitics`<br>`set_input_transition`<br>`set_load`<br>`set_port_fanout_number`<br>(In RTL budgeting, the budgeter creates this constraint when the fanout number is greater than four to force compile to use stronger driving cells on high fanout outputs. The value of the constraint is limited to fifteen. In gate budgeting, `set_port_fanout_number` specifies the exact number of external fanouts driven by the specified port.) |

Table 1-2 Constraints Generated by the Budgeter (Continued)

| Type of information | Constraint commands |
|---|---|
| Design rule constraints | `set_max_capacitance`<br>`set_max_fanout`<br>`set_max_fanout_load`<br>`set_max_transition`<br>`set_min_capacitance` |
| Timing constraints | `create_clock`<br>`create_generated_clock`<br>`group_path`<br>`set_clock_gating_check`<br>`set_clock_gating_signals`<br>`set_clock_gating_style`<br>`set_clock_latency`<br>`set_clock_transition`<br>`set_clock_uncertainty`<br>`set_critical_range`<br>(If set_critical_range is not specified in the top-level constraint file, you can set budget_generate_critical_range to true to enable the budgeter to propagate 10% of the shortest clock period in the design to multiply instantiated budgeted blocks. See "budget_generate_critical_range" on page 7-15.)<br>`set_default_input_delay`<br>`set_default_output_delay`<br>`set_false_path`<br>`set_ideal_net`<br>`set_input_delay`<br>`set_max_delay`<br>`set_max_time_borrow`<br>`set_min_delay`<br>`set_multicycle_path`<br>`set_output_delay`<br>`set_propagated_clock`<br>`set_resistance` |

*Table 1-2 Constraints Generated by the Budgeter (Continued)*

| Type of information | Constraint commands |
|---|---|
| Logic assignments | `set_case_analysis`<br>`set_equal`<br>`set_logic_dc`<br>`set_logic_one`<br>`set_logic_zero`<br>`set_opposite`<br>`set_unconnected` |

The budgeter does not automatically save the budgeted constraints to a file. To do this, use the `dc_allocate_budgets -write_script` command.

Because the `dc_allocate_budgets` command does not capture delays and parasitics annotated on internal nets of the characterized instance, the generated script does not include this information. To export this information to module-level runs, in dc_shell use the `write_sdf` command.

## Budgeting Report

The `report_path_budget` command generates the budgeting report, which shows you how the budgeter allocated delay to the timing paths. In addition to the delays, this report shows the budget type as follows:

- Fixed

  Interblock (glue) logic, don't touch logic, back-annotated wire delay; designated in reports by the budget attribute F.

- Allocated

Automatically allocated by the budgeter; designated in reports by the budget attribute A.

- User-specified

  Designated in reports by the budget attribute U.

Example 1-1 shows the budgeting report.

*Example 1-1 Budgeting Report*

```
report_path_budget -max 4 -verbose

****************************************
Report : budget
        -path full
        -delay max
        -max_paths 4
Design : top
Version: V-2004.06
Date   : Tue Mar 23 11:35:11 2004
****************************************

Attributes
-----------
U   User Specified Budgets
A   Automatic Budgets
F   Fixed delay

Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed

....
  Startpoint: u2/ff_m_reg
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: u3/ff_r_reg
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  top                10KGATES              ssc_core
  middlepipe         5KGATES               ssc_core
  rearpipe           5KGATES               ssc_core
```

```
Point                  Budget     Delay     Slack  Type
-------------------------------------------------------
clock clk (rise edge)
                         0.00      0.00      0.00  F
clock network delay      0.00      0.00      0.00  F
u2/ff_m_reg/CLK          0.00      0.00      0.00  A
u2/ff_m_reg/Q           10.00      0.37      9.63  U
u2/U4/A                  0.00      0.00     -0.00  F
u2/U4/Y                 10.00      0.10      9.90  U
u2/out_m                 0.00      0.00      0.00  F
U3/A                     0.00      0.00      0.00  F
U3/Y                     0.01      0.11     -0.10  A
u3/in_r                  0.00      0.00      0.00  F
u3/U6/A                  0.00      0.00      0.00  F
u3/U6/Y                  0.01      0.11     -0.10  A
u3/ff_r_reg/D            0.00      0.00     -0.00  F
-------------------------------------------------------
Total                   20.02      0.71     19.32
Required time            9.87      9.87
-------------------------------------------------------
Slack                  -10.15      9.17
```

For more details about the budgeting report, see "report_path_budget" on page 7-9. For examples, see "Reviewing Path Budgets" on page A-7, and "report_path_budget Example" on page B-2.

## Limitations

The budgeter has these limitations:

- There is limited support for multiply-instantiated designs.
- Timing models (Stamp and quick timing models) are considered fixed delays unless they are within a level of hierarchy. For more details, see "How Timing Models Are Budgeted" on page 5-9 and "How Setup Arcs in Timing Models Are Budgeted" on page 5-8.
- Back-annotated interblock net delays, setup times, and clock network delays are considered fixed delays.

- Budgeting supports only one operating condition and one library mode. Simultaneous min-max mode (with both minimum and maximum libraries) is not supported.
- The input and output delay constraints from the budgeter are the same for both minimum and maximum conditions.

# 2

# Budgeting Methodology

The following sections describe the budgeting methodology:

## Budgeting Mode

The type of design you have determines your budgeting mode. Table 2-1 lists the design inputs appropriate for each budgeting mode.

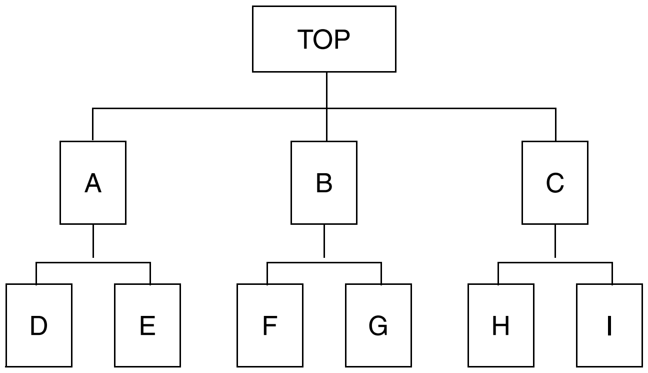
*Table 2-1 Inputs for Budgeting*

| Input | Budgeting mode |
|---|---|
| RTL source code (VHDL, Verilog), GTECH database with logical hierarchy, or GTECH netlist with logical hierarchy. Design contains very little mapped blocks. | RTL |
| RTL source code (VHDL, Verilog), GTECH database with logical hierarchy, GTECH netlist with logical hierarchy, or gate-level mapped netlists. Design typically contains a balanced mixture of RTL and mapped gates. | Mixed |
| Gate-level (mapped) netlist with logical hierarchy. Design contains very little RTL. | Gate |

## RTL Budgeting Mode Methodology

In RTL budgeting mode, only RTL is budgeted; mapped gates are treated as fixed delays. Figure 2-1 shows the hierarchical design used in the methodology description. Figure 2-2 shows the RTL mode budgeting flow.

*Figure 2-1    Hierarchical Design*



Use the following methodology for RTL budgeting mode:

1. Review the budgeting design recommendations discussed in "Designs Best Suited for Budgeting" on page 1-7 and update your design as needed. Determine which cells to budget or what hierarchy levels to budget. See "Specifying the Levels of Logic to Budget" on page 3-2.

2. Read in the design and apply the top-level constraints.

3. Set user budgets as needed. See "Setting a User Budget on a Timing Path" on page 3-10.

4. Apply the `set_dont_touch` command to cells and nets as needed. The `dc_allocate_budgets` command honors `dont_touch` attributes and passes them to the constraints written out. See "Specifying Cells as Fixed (dont_touch Logic)" on page 3-5.

5. Apply the `set_dont_touch_network` command to the clock circuitry as needed. The `dc_allocate_budgets` command honors `dont_touch_network` attributes and passes them to the constraints written out. See "How Clock Networks Are Budgeted" on page 5-7.

6. Use the `read_sdf` and `read_parasitics` commands to read in back-annotated information, as needed. See Chapter 4, “Budgeting With Back-Annotated Information.”

7. Use the `check_timing` command to identify problems or timing assertions in the design that might make the budget incomplete or inaccurate. See “How To Check the Design Before Budgeting” on page 5-5.

8. Use the `set_default_driving_cell` command to override the default choice of driving cell, as needed. By default, `dc_allocate_budgets` chooses the smallest area cell as the drive cell even if it has a `dont_use` attribute applied to it. See “Specifying a Driving Cell in RTL Budgeting” on page 3-16.

9. Run `dc_allocate_budgets -mode rtl` with the desired options. See “dc_allocate_budgets” on page 7-5 for option details.

10. Run `report_path_budget` to review the results of budgeting. See “report_path_budget” on page 7-9, “Reviewing Path Budgets” on page A-7, and “report_path_budget Example” on page B-2.

11. Compile the design bottom up using the constraints generated by the `dc_allocate_budgets` command. Use `compile -map_effort medium`.

    - Compile designs D, E, F, G, H, I, then set `dont_touch` on these compiled blocks.

    - Compile designs A, B, C, then set `dont_touch` on these compiled blocks. For examples, see “Compiling the Design With Budgeted Constraint Files” on page A-10.

12. Reapply the top-level constraints to the top-level design, TOP.

13. Compile TOP using `compile -map_effort medium`. Note that the subblocks have the `dont_touch` attribute applied.

14. Remove `dont_touch` attributes on the subblocks A, B, C, D, E, F, G, H, and I.

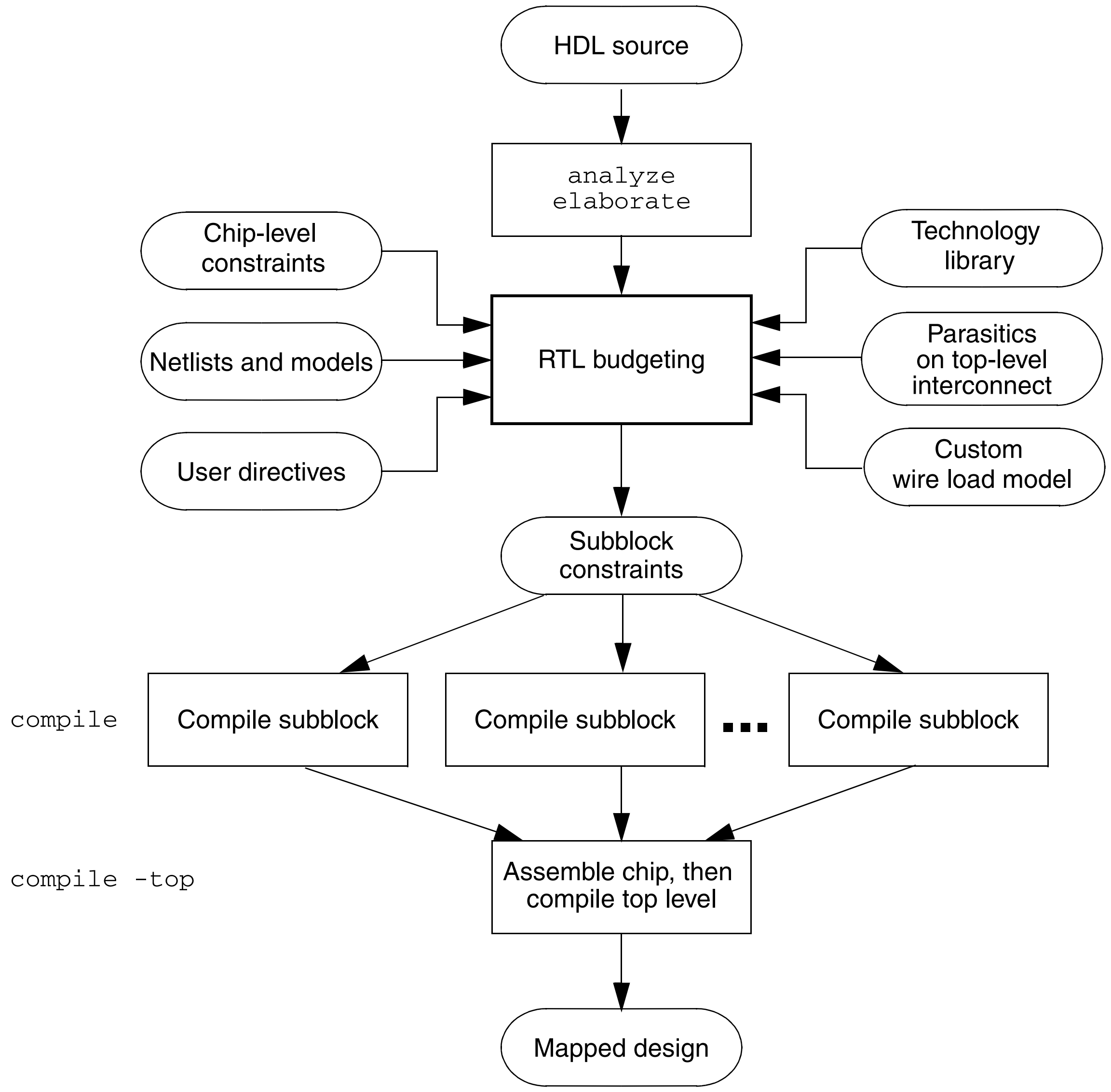
15. If there are any interblock violations after compiling TOP, fix them using `compile -top`. See "How To Resolve Interblock Timing Violations" on page 5-3.

16. Run and analyze timing reports to determine whether the design meets your goals. See Chapter 5, "Analyzing Your Budget and Solving Problems."

17. If timing or design rule violations exist, run the gate budgeting mode methodology. See "Gate Budgeting Mode Methodology" on page 2-9.

Figure 2-2 shows the RTL budgeting mode flow.

*Figure 2-2    RTL Budgeting Mode Flow*



## Mixed Budgeting Mode Methodology

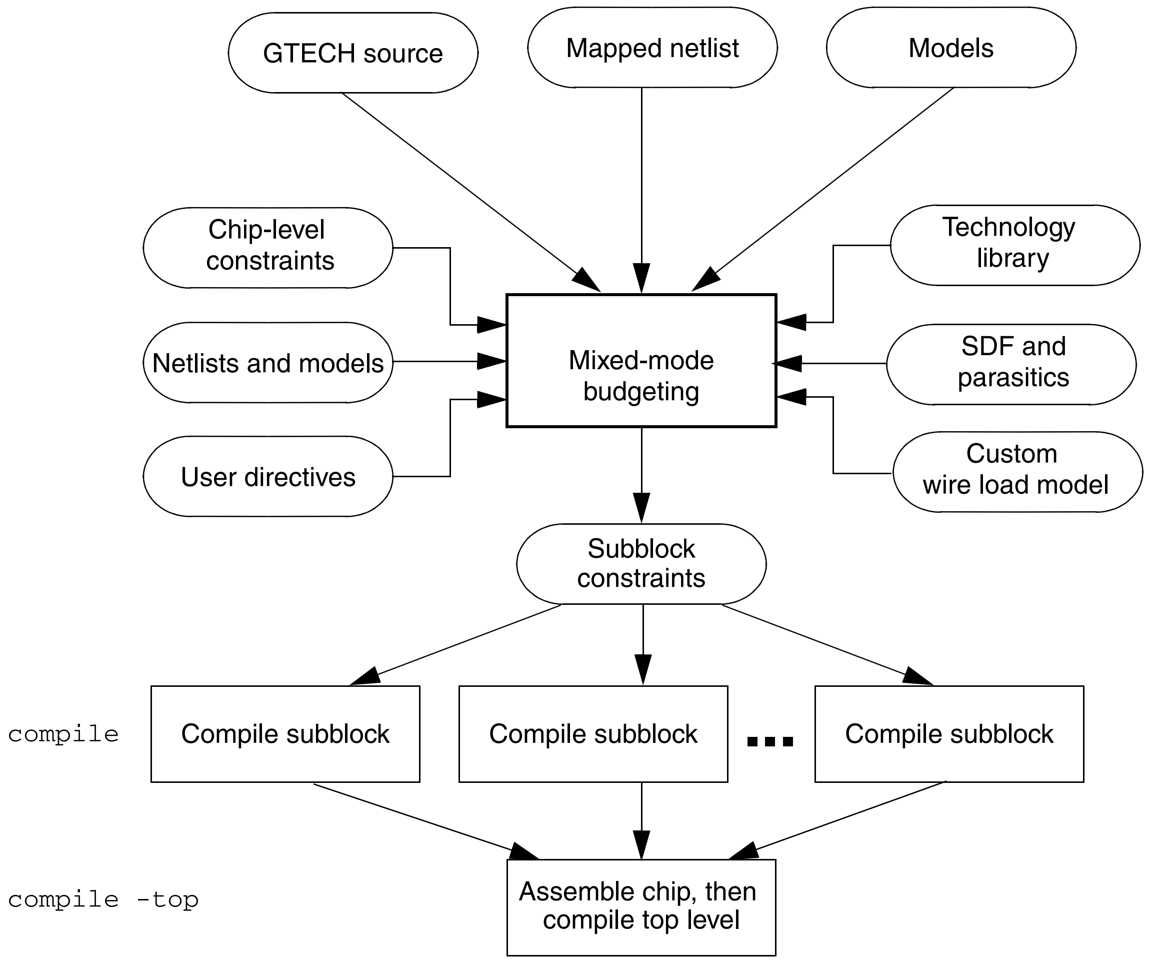
In mixed budgeting mode, both RTL and mapped gates are budgeted. Figure 2-1 on page 2-3 shows the hierarchical design used in the methodology description.

Use the following methodology for mixed budgeting mode:

1. Do steps 1–8 of the RTL budgeting mode methodology. See "RTL Budgeting Mode Methodology" on page 2-2.
2. Run `dc_allocate_budgets -mode mixed` with other options, as desired. See "dc_allocate_budgets" on page 7-5 for option details.
3. Run `report_path_budget` to review the results of budgeting.
4. Compile the design bottom up using the constraints generated by the `dc_allocate_budgets` command. For mapped designs, use `compile -incremental -map_effort medium`; for unmapped designs, use `compile -map_effort medium`.
   - Compile designs D, E, F, G, H, I, then set `dont_touch` on these compiled blocks.
   - Compile designs A, B, C, then set `dont_touch` on these compiled blocks. For examples, see "Compiling the Design With Budgeted Constraint Files" on page A-10.
5. Do steps 12–17 of the RTL budgeting mode methodology. See "RTL Budgeting Mode Methodology" on page 2-2.

Figure 2-3 shows the mixed budgeting mode flow.

*Figure 2-3 Mixed Budgeting Mode Flow*



## Gate Budgeting Mode Methodology

In gate budgeting mode, only mapped gates are budgeted; any RTL is considered fixed. Gate budgeting mode generates an incrementally mapped design.

Figure 2-1 on page 2-3 shows the hierarchical design used in the methodology description.

Use the following methodology for gate budgeting mode:

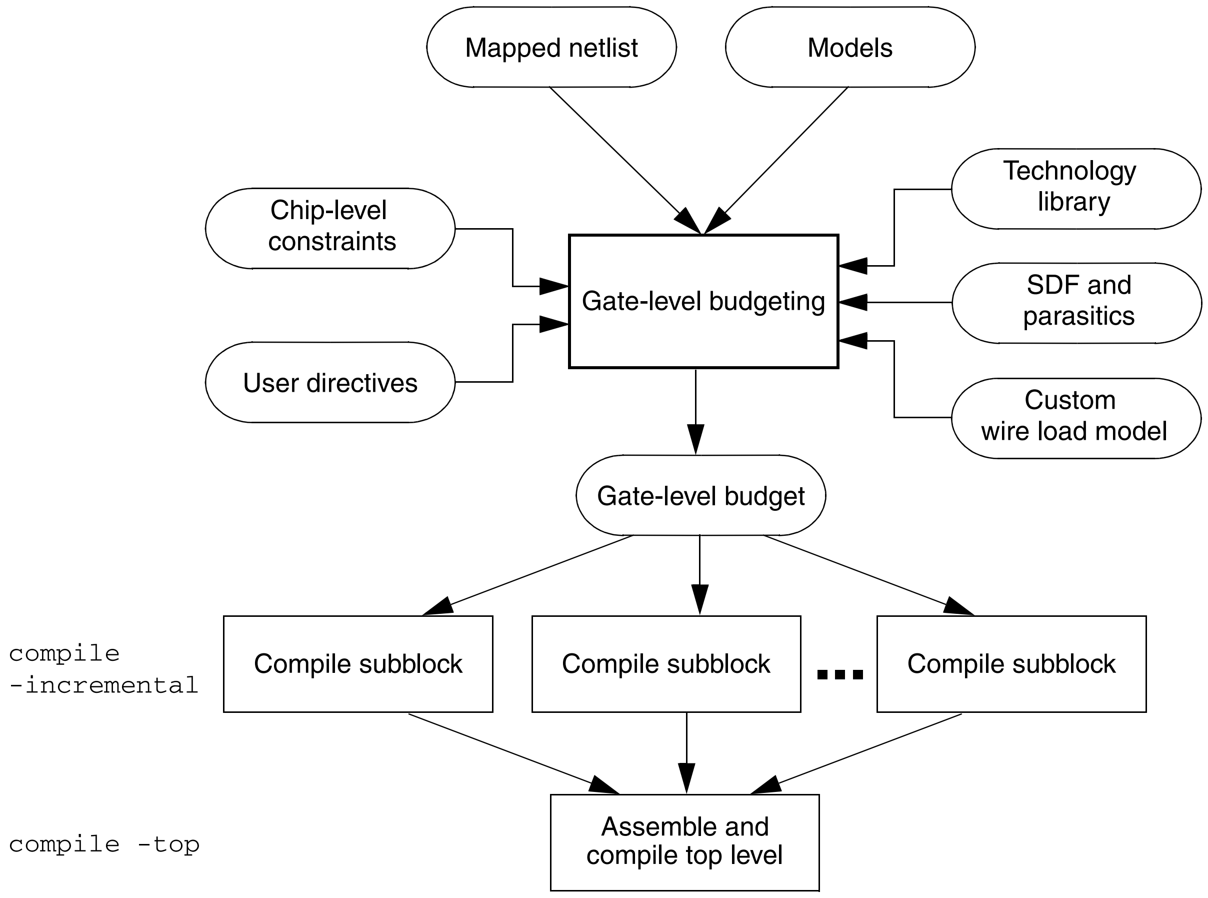
1. Do steps 1–8 of the RTL budgeting mode methodology. See “RTL Budgeting Mode Methodology” on page 2-2.
2. Run `dc_allocate_budgets -mode gate` with other options, as desired. See “dc_allocate_budgets” on page 7-5 for option details.
3. Run `report_path_budget` to review the results of budgeting.
4. Incrementally compile the design bottom up using the constraints generated from budgeting.
    - Compile designs D, E, F, G, H, I. Use `compile -incremental -map_effort medium`
    - Compile designs A, B, C. Use `compile -incremental -map_effort medium`
5. Compile TOP. Use `compile -top -map_effort medium` and the top-level constraints to fix timing and design rule violations and to resolve interblock timing violations.
6. Analyze the results to determine whether the design meets your goals. Run timing reports.
7. If the results are not what you want, refine the design.

- If timing violations are small, run gate-level budgeting again using the mapped netlist from the previous budgeting and perform `compile -incremental -map_effort high` on each block using constraints from gate-level budgeting.
- If timing violations are large, perform a full compile using the RTL and the gate-level budgets.

8. After refining the gate-level netlist, if you are not satisfied, further refine the design by repeating gate budgeting mode. See "What To Do If You Don't Meet Design Goals" on page 5-2.

Figure 2-4 shows the gate budgeting mode flow.

*Figure 2-4    Gate Budgeting Mode Flow*



# 3

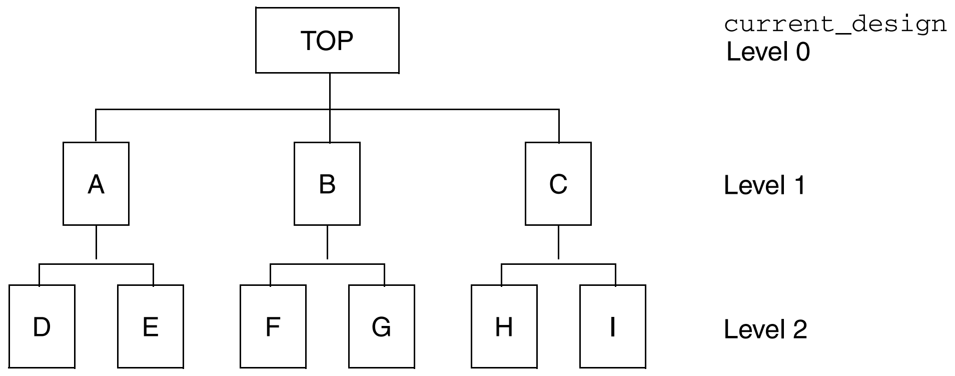
# Interactive Budgeting

The following sections describe interactive budgeting:

## Specifying the Levels of Logic to Budget

The levels of hierarchy budgeted depend on the level you specify. Figure 3-1 shows how hierarchy levels are numbered for the budgeter.

*Figure 3-1 Hierarchy Levels*



### Example

To budget blocks A, B, and C, enter

```
current_design TOP
dc_allocate_budgets -level 1
```

To budget blocks A, B, C, D... I, enter

```
current_design TOP
dc_allocate_budgets -level 2
```

## Controlling the Budgeting of Interblock Logic

By default, the budgeter considers any nonhierarchical logic between budgeted blocks as variable and allocates budgets to such logic. This logic is called interblock or glue logic. Although this logic is considered budgeted, no constraints can be generated for it.

You can direct the budgeter not to budget interblock logic by using the `dc_allocate_budgets -no_interblock_logic` command. In this case, the budgeter considers interblock logic as fixed and does not budget it.

In Figure 3-2 on page 3-4, design Ub contains interblock logic: cells U101 and U102. Design Ua contains no interblock logic. If you only want to budget the blocks Ua/Uf, Ua/Ug ,Ub/Ux, and Ub/Uz, use the following command:

```
dc_allocate_budgets -no_interblock_logic{Ua/Uf Ua/Ug Ub/Ux
Ub/Uz}
```

The budgeter does not budget the interblock logic U101 and U102 because of the`-no_interblock_logic` option or blocks Ua/Uh and Ub/Uy because these blocks are not specified in the cell list. The delays through these cells are treated as fixed delay when budgets are allocated for all other cells in the designs.

For a clock period of 10 ns, the budgeter makes the following allocations:

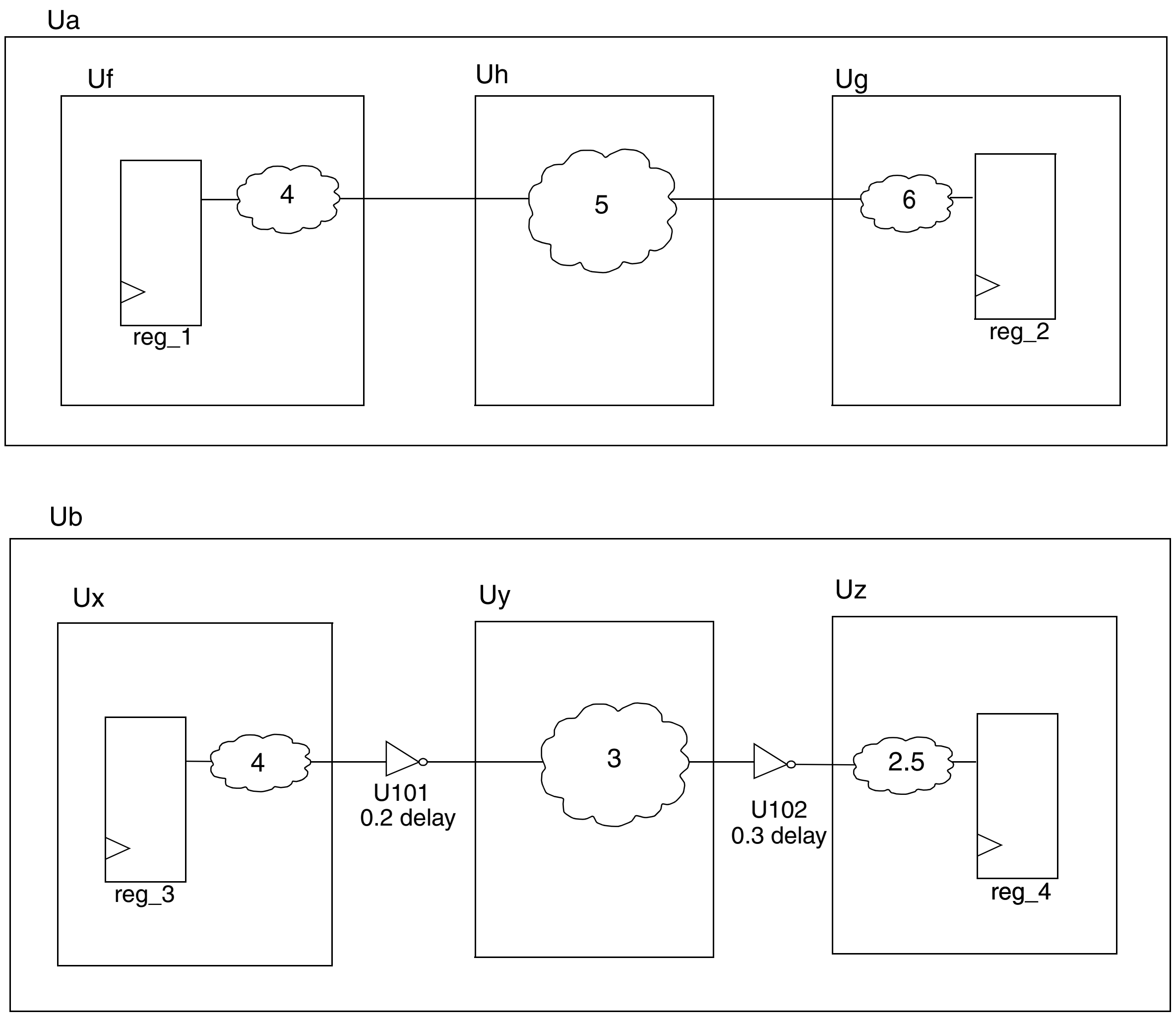
- Budget allocated for the timing path from reg_1 to reg_2 = 10 (clock period) – 5 (fixed delay in cell Ua/Uh) = 5

- Budget allocated for the timing path from reg_3 to reg_4 = 10 (clock period) – 3 (fixed delay in cell Ub/Uy) – [(0.2 + 0.3) (fixed delays in U101 an U102)] = 6.5 ns

  Note:

  For this example, internal register delays are ignored.

*Figure 3-2 Design Ua and Ub*



In general, budget interblock logic when you are following a bottom-up compile strategy.

Do not budget interblock cells when your design

- Is close to meeting timing
- Has specific interblock repeater cells inserted
- Has been laid out
- Has custom wire load models

  Note:

  Interblock nets back-annotated with SDF, Standard Parasitic Exchange Format (SPEF), or RC are considered fixed and are not budgeted.

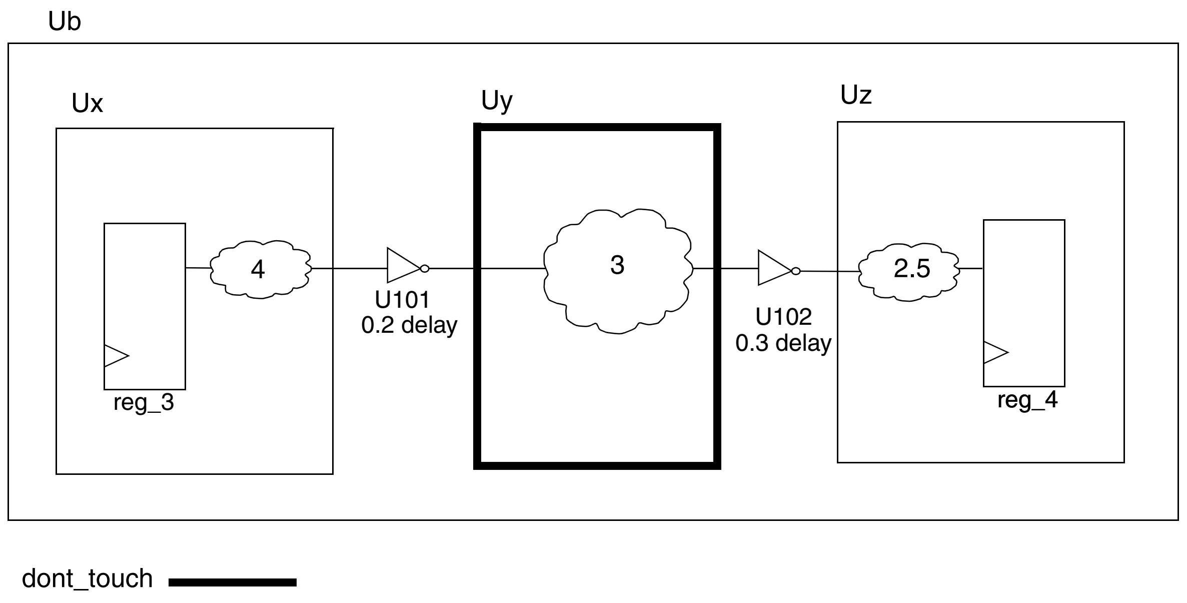
## Specifying Cells as Fixed (dont_touch Logic)

During budget allocation, the budgeter considers the delay of `dont_touch` logic inside budgeted cells as fixed.

In Figure 3-3, to apply the dont_touch attribute to Ub/Uy, use the following command:

```
set_dont_touch { Ub/Uy } "true"
```

*Figure 3-3 Design Containing dont_touch Logic*



When you allocate budgets for the Ub design, for example:

```
dc_allocate_budgets
```

Ub/Uy is treated as a fixed delay and not budgeted.

For a clock period of 15 ns, the budgeter makes the following allocation:

- Budget allocated for the timing path from reg_3 to reg_4 = 15 (clock period) – 3 (fixed delay in Ub/Uy) = 12 ns

For more information about the `dont_touch` attribute, see the man page.

## Preserving the Clock Buffer Network During Optimization

You can apply the `set_dont_touch_network` command to clocks, pins, or ports to prevent the tool from modifying these objects during optimization. The `set_dont_touch_network` command sets the `dont_touch_network` attribute on clocks, pins, or ports in the current design. The command is used primarily for clock circuitry.

When a design is optimized, synthesis assigns `dont_touch` attributes to all cells and nets in the transitive fanout of `dont_touch_network` objects. Setting `dont_touch_network` on a clock object prevents synthesis from modifying the clock buffer network.

The `dc_allocate_budgets` command honors `dont_touch_network` attributes.

### Example

To add the `dont_touch_network` attribute to the port named clock_in, enter

```
set_dont_touch_network [get_ports clock_in]
```

To remove the `dont_touch_network` attribute, use `reset_design`.

For more information about the `dont_touch_network` attribute and its effect, see the man page.

## Scaling Budgeted Constraints

You can use the `set_context_margin` command to specify a constraint margin to add to or subtract from input and output delay values when you write the context. You can specify the margin as an absolute value or as a percentage of the constraint.

Use `set_context_margin` before you run the `dc_allocate_budgets` command.

Input delays and output delays are adjusted by the margin you specify, as follows:

- To constrain more, use the default setting. The margin is added to the maximum delay values and subtracted from the minimum delay values.
- To constrain less, use the `-relax` option. The margin is subtracted from the maximum delay values and added to the minimum delay values.

The margin applies to the current design if you do not use an object list. When the budgeter determines the margin for a pin, it uses the value you specify for the pin. If you do not specify this value, the budgeter uses the value set for the parent cell. If no value is set for the parent cell, the budgeter uses the value set for the current design. If no margin is specified, the budgeter uses 0.0, the default value.

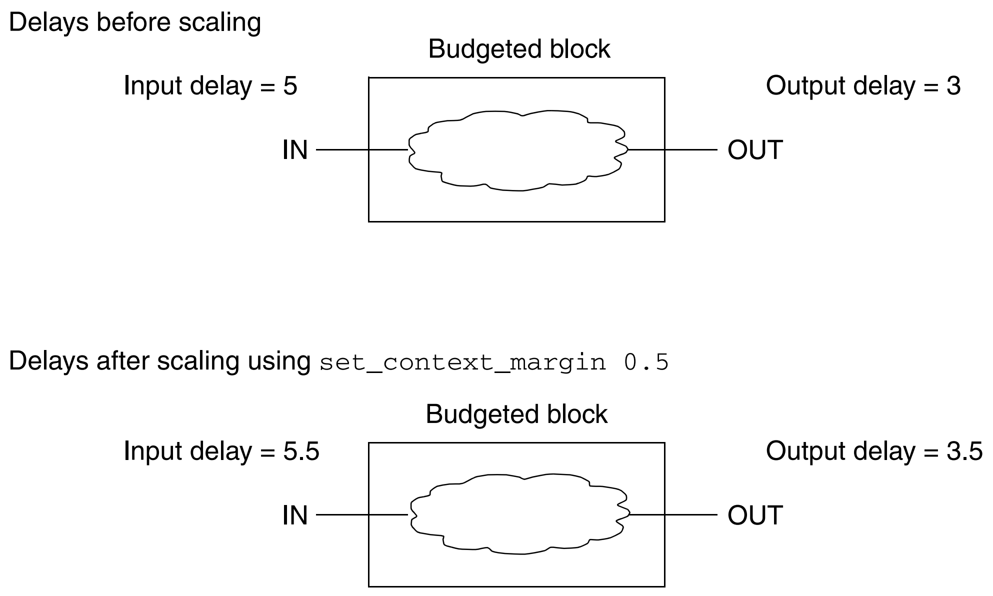
### Example 1

To add a margin of 0.5 to the maximum values of input and output delay constraints of all objects in the current design and to subtract the margin 0.5 from the minimum values of input and output delay constraints of all objects in the current design, enter

```
set_context_margin 0.5
```

Figure 3-4 shows the input delay and the output delay for a budgeted block before a margin is specified and the resulting, more restrictive, delays after a margin of 0.5 is specified for the block.

*Figure 3-4    Applying a Scaling Margin*



### Example 2

To reduce the maximum values for input delays on I2/IN by 10 percent (make the input delay values 10 percent less), enter

```
set_context_margin -relax -max \ -percent 10.0 I2/IN
```

### Example 3

To reduce the maximum values for input delays on I2/IN by 10 percent (make the input delay values 10 percent less) and write the context, enter

```
set_context_margin -relax -max -percent 10.0 I2/IN
dc_allocate_budgets -write_script
```

For detailed information, see "set_context_margin" on page 7-13 and the man page.

## Setting a User Budget on a Timing Path

The budgeter provides the following commands that allow you to set and remove a budget on a timing path:

- `set_user_budget`
- `remove_user_budget`

### Setting User Budgets

You can allocate budgets manually by using directives to the budgeter. The budgets you specify can be an absolute value or a percentage of the constraint to be allocated. The budget allocation process honors these directives.

The `set_user_budget` command specifies absolute budgets or budget allocation ratios on budgeted blocks.

- If you do not specify a budget for a budgeted cell, the budgeter performs budget allocation automatically for the cell.
- If you specify budgets for cells that are not ultimately budgeted, the budgeter ignores the budgets.

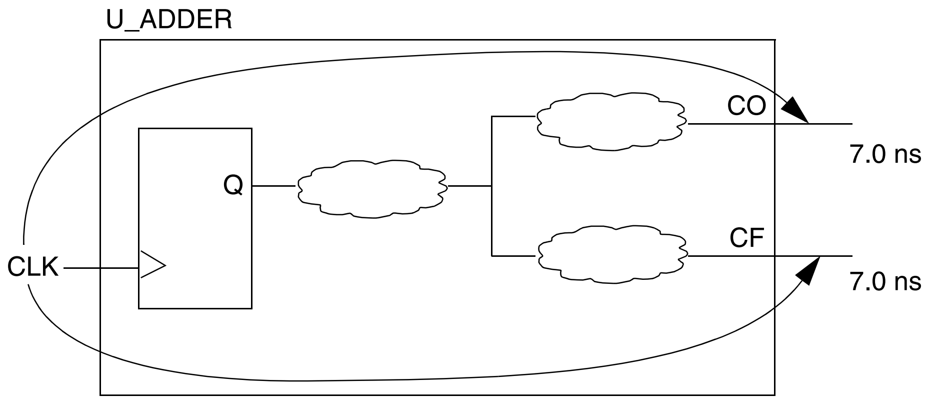
Note:

You cannot set a user budget on a back-annotated global net or any fixed delay.

For the U_ADDER cell below, this command,

```
set_user_budget -from CLK -to {U_ADDER/CO U_ADDER/OF} 7.0
```

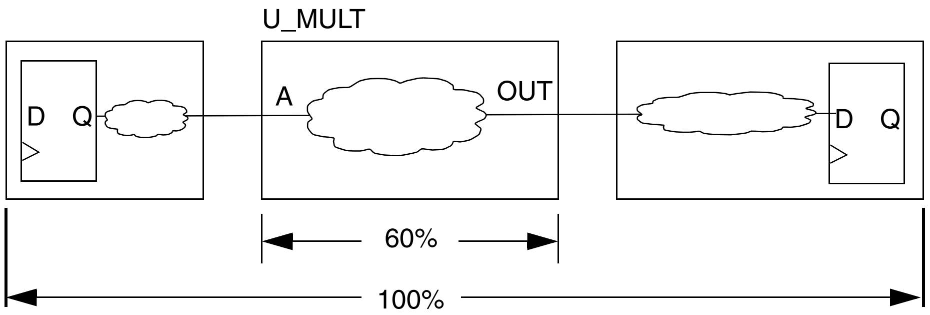
sets a budget of 7.0 from registers clocked by CLK to two outputs of the U_ADDER cell.



For the cells below, this command,

```
set_user_budget -from U_MULT/A -to U_MULT/OUT -percent 60.0
```

specifies the budget from the A input of U_MULT to the OUT output of U_MULT as 60% of the total path constraint.



For more information, see “set_user_budget” on page 7-15 and the man page.

You can remove information specified by `set_user_budget` with the `remove_user_budget` command.

### Removing User-Defined Budget Information

Use the `remove_user_budget` command to remove information you specified previously using the `set_user_budget` command.

You must specify an object list of the pins or clocks from which the budget is specified and an object list of the pins or clocks to which the budget is specified. For example, to remove user-specified budget information, from clock clk1 to the two outputs, enter

```
remove_user_budget -from clk1 -to {I2/out1 I2/out2}
```

For more information, see “remove_user_budget” on page 7-9 and the man page.

## Setting Minimum Budgets on All Timing Paths

The `dc_allocate_budgets` command has the following options that enable you to specify a minimum delay on timing path budgets:

```
dc_allocate_budgets

         [-min_register_to_output minimum_budget]
         [-min_input_to_output minimum_budget]
         [-min_input_to_register minimum_budget]
```

Use these options carefully because they can cause timing constraint violations (fixed delays exceed timing constraints).

To set the minimum register to output budget to be 0.45, use the following command:

```
dc_allocate_budgets -min_register_to_output 0.45
```

Example 3-1 shows budget report before and after using the command.

Because the timing path from the register, I_DATA_PATH/Oprnd_B_reg[0]/Q, is directly connected to the output port, Oprnd_B[0], of the I_DATA_PATH block, the clk-Q delay for this path (I_DATA_PATH/Oprnd_B_reg[0]/Q) is changed from 0.21 to 0.45 as a result of setting the minimum budget.

*Example 3-1 Budget Reports Before and After Setting a Minimum Budget*

```
****************************************
Report : budget
        -path full
        -delay max
        -max_paths 1
Design : RISC_CORE
Version: V-2004.06-LS-BETA2
Date   : Tue Mar 23 12:56:35 2004
****************************************


Attributes
-----------
U   User Specified Budgets
A   Automatic Budgets
F   Fixed delay


Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed


  Startpoint: I_DATA_PATH/Oprnd_B_reg[0]
              (rising edge-triggered flip-flop clocked by my_clock)
  Endpoint: I_ALU/Zro_Flag_reg
            (rising edge-triggered flip-flop clocked by my_clock)
  Path Group: my_clock
  Path Type: max


  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISC_CORE          10KGATES              ssc_core
  ALU                5KGATES               ssc_core
  ALU_DW01_sub_16_0  5KGATES               ssc_core


  Point                  Budget     Delay     Slack  Type
  -------------------------------------------------------
  clock my_clock (rise edge)
                           0.00      0.00      0.00  F
  clock network delay      0.00      0.00      0.00  F
  I_DATA_PATH/Oprnd_B_reg[0]/CLK
                           0.00      0.00      0.00  A
  I_DATA_PATH/Oprnd_B_reg[0]/Q
                           0.21      0.35     -0.14  A
  I_DATA_PATH/Oprnd_B[0]
                           0.00      0.00      0.00  F
  I_ALU/Oprnd_B[0]         0.00      0.00      0.00  F
  I_ALU/Zro_Flag_reg/D
                           3.15      3.77     -0.62  A
  -------------------------------------------------------
  Total                    3.36      4.13     -0.76
  Required time            3.72      3.72
  -------------------------------------------------------
  Slack                    0.35     -0.41


****************************************
Report : budget
        -path full
        -delay max
        -max_paths 1
Design : RISC_CORE
Version: V-2004.06-LS-BETA2
Date   : Tue Mar 23 12:59:36 2004
****************************************


Attributes
-----------
U   User Specified Budgets
A   Automatic Budgets
F   Fixed delay


Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed
```

```
Startpoint: I_DATA_PATH/Oprnd_B_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: I_ALU/Zro_Flag_reg
          (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max


Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
RISC_CORE          10KGATES              ssc_core
ALU                5KGATES               ssc_core
ALU_DW01_sub_16_0  5KGATES               ssc_core


Point                     Budget     Delay     Slack  Type
------------------------------------------------------
clock my_clock (rise edge)
                            0.00      0.00      0.00  F
clock network delay         0.00      0.00      0.00  F
I_DATA_PATH/Oprnd_B_reg[0]/CLK
                            0.00      0.00      0.00  A
I_DATA_PATH/Oprnd_B_reg[0]/Q
                            0.45      0.35      0.10  A
I_DATA_PATH/Oprnd_B[0]
                            0.00      0.00      0.00  F
I_ALU/Oprnd_B[0]            0.00      0.00      0.00  F
I_ALU/Zro_Flag_reg/D
                            3.08      3.77     -0.69  A
------------------------------------------------------
Total                       3.53      4.13     -0.59
Required time               3.72      3.72
------------------------------------------------------
Slack                       0.18     -0.41
```

For option details, see “dc_allocate_budgets” on page 7-5 and the man page.

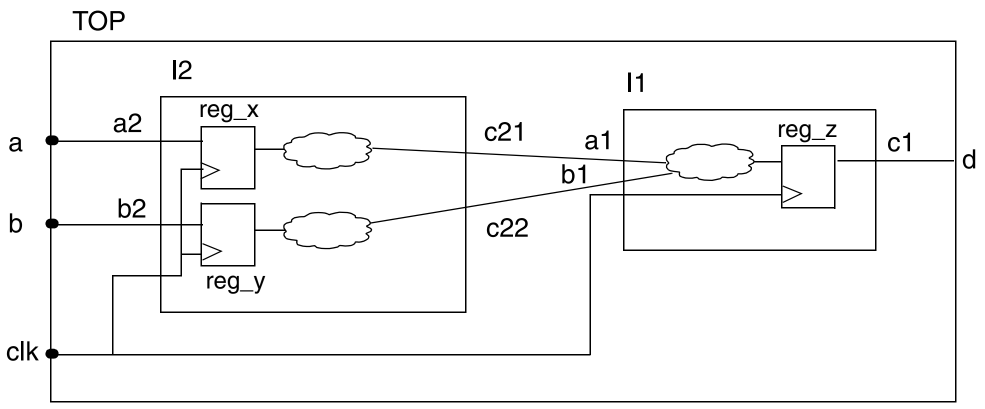
## Specifying a Driving Cell in RTL Budgeting

The algorithm used to determine the default driving cell for budgeting is based on area minimization; this typically results in a weak driving cell. If you need to change the driving cell, use the `set_default_driving_cell` command. Example 3-2 uses the `set_default_driving_cell` command to specify ff01d3, ff01d5, and ff01d7 as drivers for reg_x, reg_y, and reg_z in design TOP (Figure 3-5).

*Example 3-2 set_default_driving_cell*

```
current_design TOP
set_default_driving_cell -lib_cell ff01d3 I2/reg_x;
set_default_driving_cell -lib_cell ff01d5 I2/reg_y;
set_default_driving_cell -lib_cell ff01d7 I1/reg_z;
...
...source top-level constraints...
...run budgeter...
```

*Figure 3-5 Design TOP*



For more details about the `set_default_driving_cell` command, see the man page.

# 4

# Budgeting With Back-Annotated Information

This chapter contains the following sections:

## Budgeting With Back-Annotation During Design Exploration

You can perform budgeting with back-annotation during the exploration stages of your design. The Synopsys links-to-layout methodology recommends doing floorplanning in the early stages of your design cycle. After floorplanning is done, timing and parasitic information is extracted from the floorplan and used within Synopsys synthesis tools to generate custom wire load models.

In addition, you can generate budgets for your logical blocks using this back-annotation data. These generated constraints better reflect the delay and load information of the global routes of the design. Resynthesize the netlist using the custom wire load models and the constraints generated through budgeting based on the back-annotation data.

## Budgeting With Back-Annotation

If timing violations exist after placement and routing, you typically correct them by modifying the floorplan or the placement or routing parameters. In some cases, you might want to rebudget and reoptimize the logic of some blocks, keeping the original floorplan and block placement.

Budgeting takes into account back-annotated information such as SDF or parasitics in such a flow. For more accurate results from gate-level budgeting, annotate SDF instead of parasitics.

Note:

Interblock nets back-annotated with SDF, SPEF, or RC are considered fixed and are not budgeted.

## Guidelines for Budgeting in the Post-Layout Flow

When you allocate budgets in the post-layout flow, the budgeter

- Considers as fixed the net delays and does not attempt to change the budgets of global net delays
- Does not consider as fixed the cell delays and parasitics that are back-annotated inside budgeted blocks

## Timing Information Considered During Allocation

The budgeter can consider timing information from layout tools during budget allocation. This timing information includes

- Standard Delay Format (SDF) files
- Lumped RC (resistance and capacitance) data
- RSPF (Reduced Standard Parasitic Format) data
- DSPF (Detailed Standard Parasitic Format) data
- SPEF (Standard Parasitic Extended Format) data

For example, placement and routing might cause long nets, increasing the net delays over the statistical wire load models. Back-annotating global net delays or parasitics will modify the budgets created.

The `read_sdf` and `read_parasitics` commands annotate this information. For information about these commands, see the man pages.

## Reading RSPF, DSPF, and SPEF Parasitics Files

You can use the RSPF, DSPF, and SPEF parasitics files in the budgeting flow.

The tool supports both Arnoldi and Elmore delay calculations. To specify Arnoldi calculations, use

```
read_parasitics -arnoldi
```

To specify Elmore delay calculations, use

```
read_parasitics -elmore
```

If you have multiple parasitic information, add either `-elmore` or `-arnoldi` to the last `read_parasitics` *only*, as shown in Example 4-1; this saves runtime.

*Example 4-1 Using Arnoldi Delay Calculations*

```
current_design A
read_parasitics A1.spef
read_parasitics A2.spef
read_parasitics A3.spef -arnoldi
```

The `-elmore` option makes the tool create an RC tree and back-annotate delay estimates. The delays are computed out of the parasitics information based on the Elmore delay model.

The -arnoldi option makes the tool create an RC tree and back-annotate delay estimates. The delays are computed out of the parasitics information via the Arnoldi delay calculator.

If neither `-elmore` nor `-arnoldi` is specified, the RC tree is created without estimating and back-annotating delays.

**Important:**

Do not use any command between the `read_parasitics` commands because the parasitic data from the first `read_parasitics` command will be lost. In Example 4-2, the A1.spef data is lost because the `report_timing` command is used between the `read_parasitics` commands.

*Example 4-2 Lost Parasitics Data*

```
current_design A
read_parasitics A1.spef
report_timing
read_parasitics A2.spef
read_parasitics -elmore A3.spef
```

Use the `read_parasitics` command to read in RSPF, DSPF, and SPEF parasitics files and annotate the current design. You do not need to specify the format because the reader automatically detects it.

Net and instance pin names in the design must match the names in the parasitics file. For example, if you create the parasitics file from a design using VHDL naming conventions, the design name must use VHDL naming conventions.

When parasitic files are read in, the default assumption is that capacitances specified in the SPEF files do not include the pin capacitances. The pin capacitances used are the values specified in the Synopsys libraries of the design. The pin capacitances specified in SPEF are ignored.

The reduced and detailed RC networks specified in SPEF files are used to compute effective capacitances dynamically during delay calculation. The capacitances reported with most report commands (for example, `report_timing` and `report_net`) are the lumped capacitance, also known as Ctotal. Ctotal is the sum of all capacitance of a net as specified in the SPEF, to which pin capacitances are also added.

The parasitics files must be in SPEF IEEE 1481-1997, RSPF, or DSPF.

For more information and the syntax, see the `read_parasitics` man page.

#### Example

To read in the parasitics for the U1/U2 path within a file called sub_design1.rspf using the RSPF file format, enter

```
read_parasitics -elmore design1.rspf
```

### Removing Annotated Parasitics

Use `remove_annotated_delay`, `remove_annotated_transition`, or `remove_attribute load` to remove annotated parasitics and delays on nets of the current design.

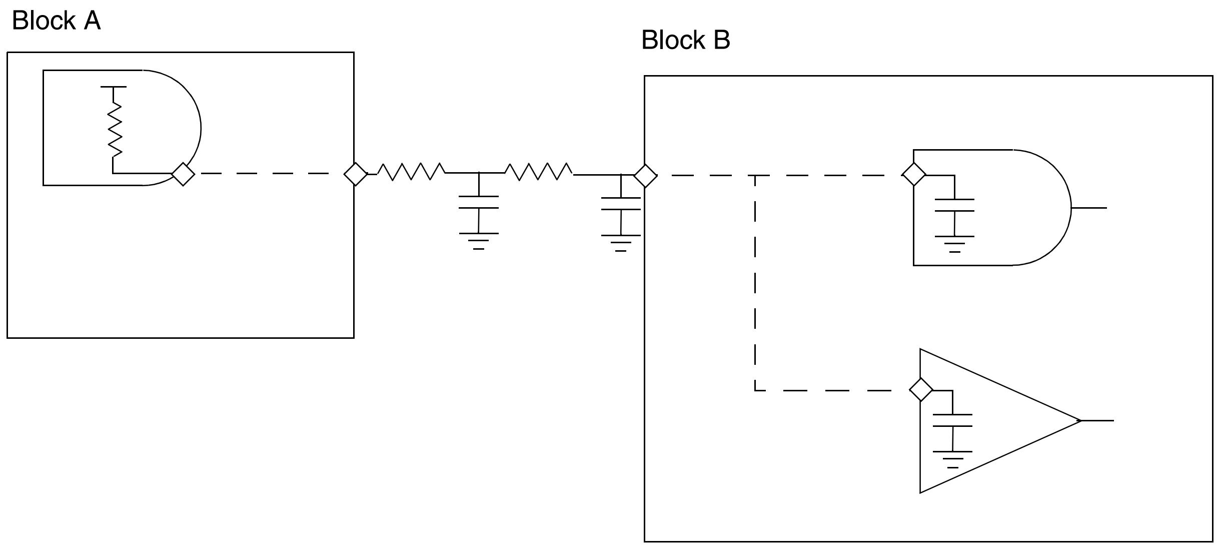
The `reset_design` command also removes parasitics read and annotated with `read_parasitics`.

## Completing Incomplete Back-Annotation

In a budgeting flow with RTL or black box planning, only top-level routing might be complete and detailed parasitics might be present between pins of top-level blocks after floorplanning. However, inside the blocks there might not be routing information, resulting in the top-level routing information being discarded because it does not extend to leaf pins. By default, incomplete RC network information (SPEF, DSPF, or RSPF information that does not extend to leaf-level cell pins) is ignored during timing analysis or budget allocation.

Figure 4-1 shows an incomplete RC network.

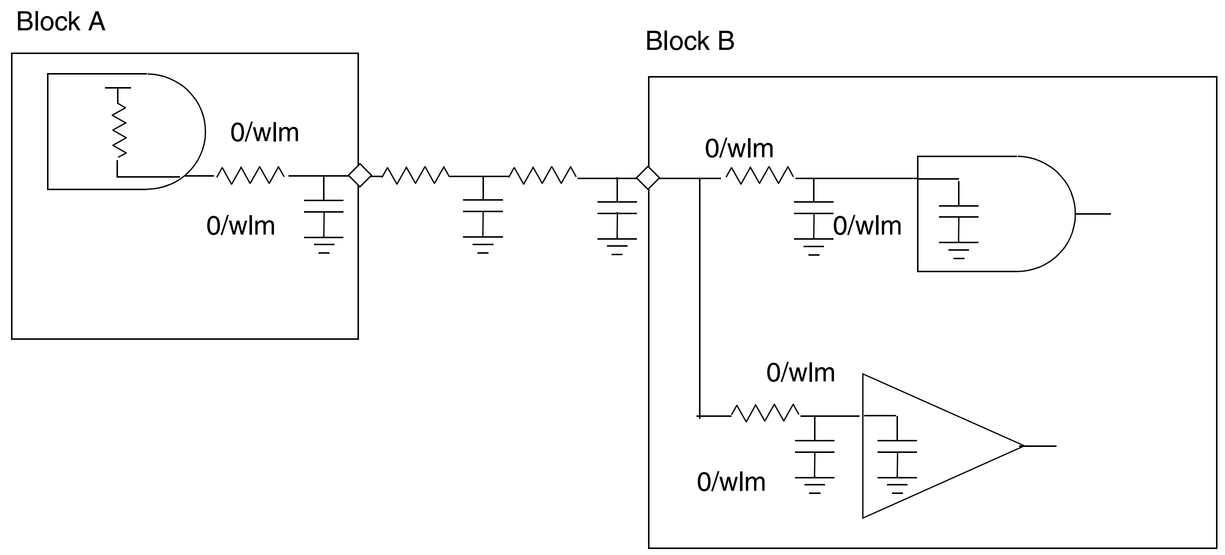
*Figure 4-1 Incomplete RC Network*



## Completing RC Networks

Budgeting takes into account interblock net delays with detailed parasitics if the parasitics are complete. Use the `read_parasitics -complete_with` command or the `complete_net_parasitic -complete_with` command to extend incomplete detailed parasitics to leaf pins using shorts (a resistor of value 0.0 or RC values derived using wire load models) When you use `-complete_with`, detailed parasitics on partially annotated interblock nets are completed, as shown in Figure 4-2, so that budgeting and timing can use them to compute more accurate budgets and delays.

*Figure 4-2 Complete RC Network*



When you use the `-complete_with` option, keep the following points in mind:

- RC network completion allows the budgeter to use incomplete RC network information by completing missing portions of back-annotated RC networks (SPEF, DSPF, or RSPF) so that they extend up to the leaf-level cell pins.
- Missing portions of RC networks can be completed with either of the following values:
  - 0 values for missing R and C (use `-complete_with zero`)
  - R and C values derived from wire load models (use `-complete_with wlm`)

- RC network completion is useful in early design phases to generate more accurate budgets with detailed RC information from floorplanning (top-level routing).

### Example

To complete detailed net parasitics, use a command sequence like this:

```
read_verilog netlist.v
current_design top_design
link
source top_constraints.pt
...
read_parasitics -elmore -complete_with wlm
top_interconnect.dspf
...
dc_allocate_budgets ...
...
```

For complete information, see the `read_parasitics` and `complete_net_parasitics` man pages.

## SPEF IEEE 1481-1997 File Example

Example 4-3 is a part of an SPEF IEEE 1481-1997 file.

*Example 4-3    Sample SPEF IEEE 1481-1997 File*

```
*SPEF "IEEE 1481-1997"
*DESIGN "arcadia"
*DATE ""
*VENDOR ""
*PROGRAM ""
*VERSION ""
*DESIGN_FKOW ""
*DIVIDER /
*DELIMITER :
*BUS_DELIMITER []
*T_UNIT 100PS
```

```
*C_UNIT 1.00 FF
*R_UNIT 1.00 OHM
*L_UNIT 100 HENRY

*NAME_MAP
*123 inv1a2

*GROUND_NETS VSS

*D_NET test_net 727.993
*CONN
*I o/m1/i:A I
*I o/m1/inp O
*CAP
1 o/m1:inp VSS 1.0
2 test_net:1 VSS 1.0
3 test_net:2 VSS 1.0
4 o/m1/i:A VSS 1.0
*RES
1 o/m1:inp test_net:1 1.0
2 test_net:1 test_net:1 1.0
3 test_net:2 o/m1/i:A 1.0
*END
...
```

# 5

# Analyzing Your Budget and Solving Problems

This chapter contains the following sections:

## What To Do If You Don’t Meet Design Goals

If you don’t meet your design goals and you want to run an additional budget and compile pass, consider the following:

- Scale the budgeted constraints to be more aggressive (for example, 10 percent) using the `set_context_margin` command. Set the scaling margin before writing the constraints to use for the second pass compilation. (See “Scaling Budgeted Constraints” on page 3-8.)
- Isolate the blocks that contain more significant timing problems and perform the budgeting run on these blocks.
- Increase the compile effort level.
- Use compile techniques such as flattening or structuring.

## Fixing Poor Timing QoR In Lower-Level Blocks

If your design contains multiply instantiated blocks and shows poor timing QoR because of timing results in lower-level blocks, try setting the `budget_generate_critical_range` variable to true (default is false). When set to true, this variable enables the budgeter to propagate 10% of the shortest clock period in the design to multiply instantiated budgeted blocks. During a bottom-up compile flow, lower-level blocks are marked dont_touch, which prevents their violations from being fixed later in the flow by the final ‘compile –top’. By setting `budget_generate_critical_range` to true, you force compile to put more effort in optimizing critical and sub-critical paths across lower-level blocks. You may see a slight increase in runtime when this variable is true. For details, see the man page.

Note:

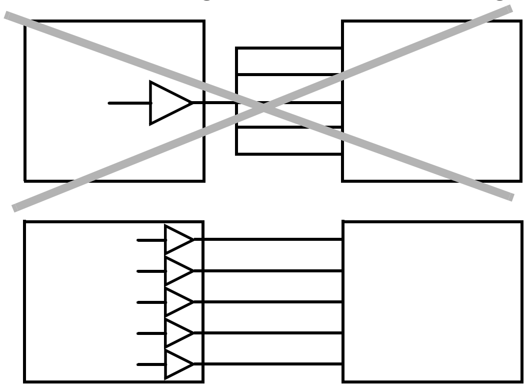
If you use `set_critical_range` in the top-level constraint file, this value is propagated to all the subdesigns and overrides the budgeter generated `set_critical_range` constraints. For variable details, see the man page.

## How To Resolve Interblock Timing Violations

After you budget and compile each block and assemble the design, you need to verify the timing of the top-level design. Some strategies to fix chip-level timing problems are

- Minimize interblock timing violations by following a design methodology in which each output of a block drives one input. This methodology avoids the need for inserting buffers on the top-level net, as shown in Figure 5-1.

*Figure 5-1 Minimizing Interblock Timing Violations*

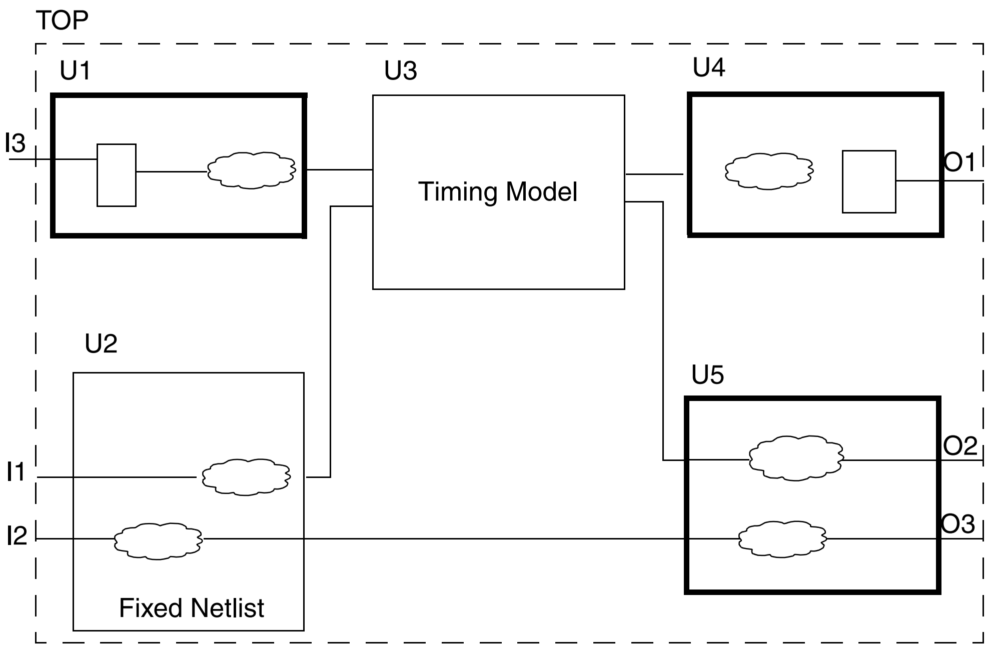


- Isolate parts of the design containing large violations and perform an incremental compile on these blocks. In this case, apply the appropriate `dont_touch` directives on the interface logic to ensure that the parts of the design that meet timing are not affected.

- Perform another budget compile iteration using the results from the first iteration to allocate more refined budgets. In this iteration, budget only critical blocks in which timing problems are most serious. (Exclude noncritical blocks by not budgeting them.) Performing timing analysis at the chip level helps identify the critical blocks.

Figure 5-2 shows a design example followed by the commands to budget the design.

*Figure 5-2 Design Example for Budgeting*



The design in Figure 5-2 contains five blocks: U1, U2, U3, U4, and U5.

- U1, U4, and U5 are synthesizable blocks.
- U2 is a fixed gate-level netlist that will not be synthesized. Set the `dont_touch` attribute on this block.

- U3 is a timing model. The budgeter reads the timing for this block but does not allocate a budget for this block.

Use the following commands to allocate budgets for synthesizable blocks U1, U4, and U5 and to generate constraints for them.

```
set_dont_touch U2
dc_allocate_budgets -write_script {U1 U4 U5}
```

## How To Check the Design Before Budgeting

Before allocating budgets, use the `check_timing` command to help identify problems or timing assertions in the design that might make the budget incomplete or inaccurate. This command generates warning and information messages about the potential problems in the design.

Note:

When you change your design or its timing assertions, reissue the `check_timing` command before you attempt budgeting.

The syntax for the `check_timing` command is

```
check_timing [-overlap_tolerance minimum_distance]
```

For more information about `check_timing`, see the man page.

Use the `report_timing_requirements` command to check the timing exceptions applied to the design. For example, use `report_timing_requirements -ignored` to list the timing exceptions set on the current design that are ignored.

Table 5-1 describes problems that might occur in a design.

*Table 5-1 Potential Design Problems*

| **Potential problem** | **Report results** |
|---|---|
| Unconstrained endpoints | Shows unconstrained register data pins or primary outputs that are not marked as false paths. |
| No clock fanin | Shows register clock pins that cannot be reached by a clock signal. Data pins on these same registers often also appear as unconstrained endpoints. |
| Multiple clock fanin | Shows register clock pins that are reached by more than one clock signal (see Figure 5-3). |
| Master-slave clock separation | Shows overlapping clocks on master-slave registers. Designs that contain master-slave latches with separate master and slave clock inputs often have a strict timing separation value between those clocks. This value ensures that the master and slave clocks do not overlap. If an overlap occurs, the master and slave latches become transparent. You can specify a minimum separation between the clocks. |
| Combinational feedback loops | Shows the combinational feedback loops, which are considered to be asynchronous. You can break loops by disabling timing arcs. The PrimeTime tool automatically breaks them. |
| Ignored attributes | Shows ignored timing exceptions. Timing exceptions can be ignored if another exception has higher precedence or if the exception was not applied to a valid path. |
| Unmapped cells | When a design contains unmapped cells (generic logic), the timing of paths through generic cells is inaccurate because generic cells have zero delay. |
| Gated clocks | Identifies gated clocks and issues a warning if the gating timing arcs are not disabled for ideal clocks (see Figure 5-4). |

*Figure 5-3 Multiple Clock Fanin*

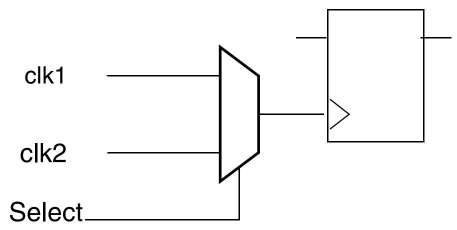
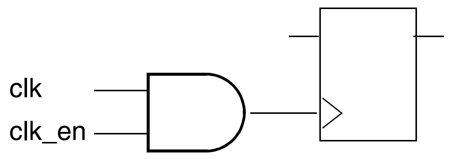


*Figure 5-4 Gated Clock*



## How Clock Networks Are Budgeted

When budgeting clock networks, the budgeter

- Propagates clock uncertainties and latencies to subblocks (The clock latency outside a block is modeled using the `set_clock_latency -source` command.)
- Generates input and output delay constraints that take into account the clock network delays in other blocks
- Considers clock network delays as fixed delays

## How Setup Arcs in Timing Models Are Budgeted

Combinational logic delays from input ports to registers are modeled as setup timing arcs in quick timing models (QTMs) and extracted timing models (ETMs). These delays are considered fixed delays even when they are in a hierarchical block. Figure 5-5 shows how the input-to-register combinational logic delay in Figure 5-6 is modeled as a setup arc.

*Figure 5-5 Input-to-Register Combinational Logic Delay*

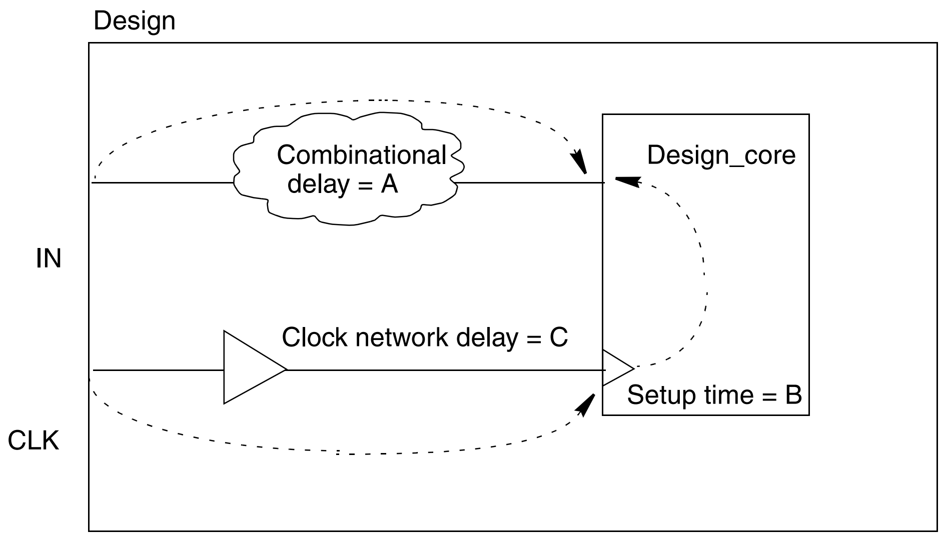
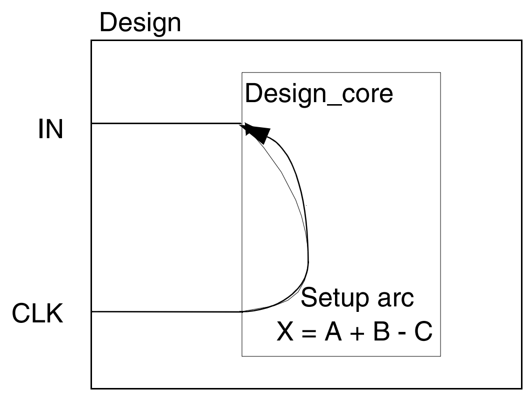


*Figure 5-6 Input-to-Register Combinational Delay Is Modeled as a Setup Arc*

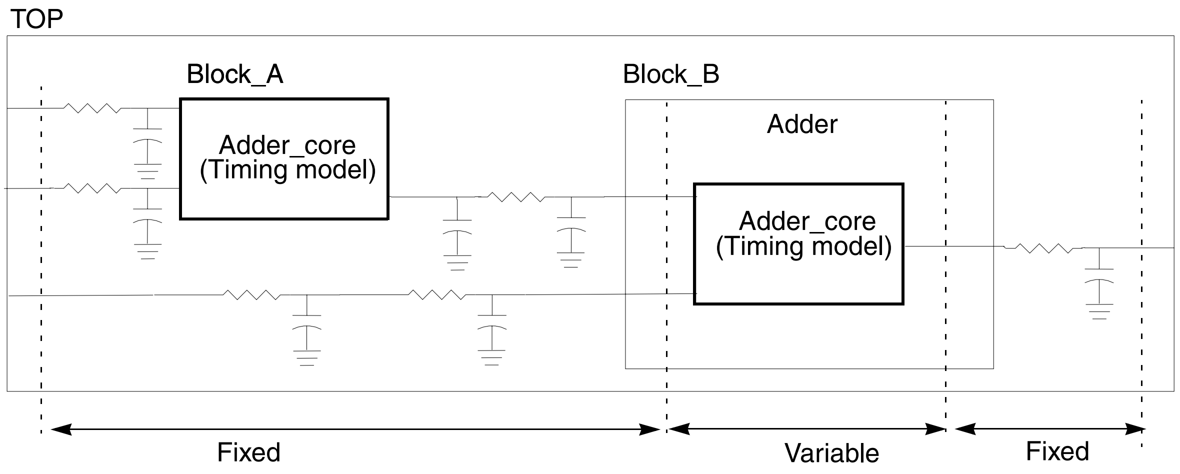


## How Timing Models Are Budgeted

The budgeter treats timing models the same as other leaf-level cells in a technology library. The term “timing model” refers to “extracted timing models” (ETMs), “quick timing models” (QTMs), or any timing model from other tools represented in Stamp format (STMs). Timing models are budgeted only if they are at a lower hierarchical level than the level at which budgeting is performed. If they are at the same level, the model is considered a fixed delay.

In Figure 5-7, if you run `dc_allocate_budgets` with the `current_design` as TOP, the timing model Block_A is considered a fixed delay component. The other instance of the same timing model within a hierarchical block, Block_B, is considered a variable delay component. All the interconnect delays, in this example, are assumed to be back-annotated RC or SDF delays.

*Figure 5-7    Timing Models With Variable and Fixed Delay Components*



Extracted timing models from PrimeTime have hierarchical wrappers. When you read these models (with hierarchical wrappers), they are not considered fixed delay components.

You can set a `dont_touch` attribute on timing models within a hierarchical block to treat them as fixed delays during budget allocation. With `dont_touch` attributes set on them, delays in timing models are considered fixed delays and subtracted from available time before budget allocation.

# 6

# Budgeting in Automated Chip Synthesis

Automated Chip Synthesis uses the Design Compiler budgeter to generate partition constraints for both GTECH and mapped designs.

This chapter describes the following sections:

## Invoking the Budgeter in Automatic Chip Synthesis

The budgeter is automatically invoked when you use the following Automated Chip Synthesis commands:

- `acs_compile_design`

  This command automatically runs `dc_allocate_budgets -mode rtl $sub_instance_list`. It uses `derive_constraint -budget` to initiate RTL budgeting. If you choose to use top-down environment propagation (TDEP) instead of RTL budgeting, only `derive_constraint` runs. See the *Automated Chip Synthesis User Guide* for more information.

- `acs_refine_design`

  This command automatically runs `dc_allocate_budgets -mode gate <instance_list>`

- `acs_recompile_design`

  This command automatically runs `dc_allocate_budgets -mode gate <instance_list>`

## Modifying Budgeting Scripts for Automatic Chip Synthesis

Budget files are created for each partition in accordance with Automated Chip Synthesis protocol. If you want to customize the budgeting script—for example, by specifying a margin to add to the input delay—you need to modify the budgeting script by using the commands discussed in Chapter 7, “Commands Specific to Budgeting for Synthesis.” For a summary of the commands, see Table 7-1 on page 7-2.

# 7

# Commands Specific to Budgeting for Synthesis

These sections describe commands used in budgeting:

## Summary of Commands Specific to Budgeting for Synthesis

Table 7-1 summarizes commands specific to budgeting for synthesis.

*Table 7-1 Summary of Commands Specific to Budgeting for Synthesis*

| Command | Description |
|---|---|
| `budget_generate_critical_range` | When set to true, the `budget_generate_critical_range` variable enables the budgeter to propagate 10% of the shortest clock period in the design to multiply instantiated budgeted blocks. Default value is false. If `set_critical_range` is specified in the top-level constraint file, this value overrides the budgeter generated `set_critical_range` constraints and is propagated to all the subdesigns. For variable details, see the man page. |
| `check_budget` | Determines whether the budgets are consistent with path constraints. Inconsistencies (constraint violations) exist when the sum of the user-specified budgets and fixed delays along a path exceeds the value of the path constraints. |
| `dc_allocate_budgets` | Allocates budgets among specified cells or pins and to blocks contained within other blocks (hierarchical blocks). |
| `set_context_margin` | Specifies a margin to add to or subtract from input and output delay values generated by the `dc_allocate_budgets -write_script` option. |
| `set_user_budget` | Specifies user directives for budget allocation. |
| `remove_user_budget` | Removes user-specified budget information previously set using the `set_user_budget` command. |
| `report_path_budget` | Generates reports that provide budget distribution, delays, budget slack, and constraints for budgeted paths. |

## check_budget

After you specify budgets and fixed delays, check the budgets for possible problems. When fixed delays exist between blocks and user-specified budgets have been set, use the `check_budget` command before you allocate budgets to determine whether the budgets are consistent with path constraints. Inconsistencies (constraint violations) exist when the sum of the user-specified budgets and fixed delays along a path exceeds the value of the path constraints.

The `check_budget` command checks automatic and user-defined or modified budgets. By default, the report shows only the number of paths that violate the timing constraints. To get detailed information, use the `-verbose` option.

The syntax is

```
check_budget [-verbose] [-tolerance tolerance]
             [-from object_list] [-to object_list]
             [-no_environment] [-no_interblock_logic]
             [cell_list]
```

| **Argument** | **Description** |
|---|---|
| `-verbose` | Shows the path segments that are causing timing constraint violations. If you omit this option, only the number of violating paths is shown. |
| `-tolerance` *`tolerance`* | Checks and reports only paths with a slack greater than the specified tolerance. If you omit this option, all paths are checked. |
| `-from` *`object_list`* | Shows only the paths from the named pins, ports, or startpoints clocked by named clocks. If you omit this option, all the paths that violate the timing constraints are shown. |

| Argument | Description |
|---|---|
| `-to` *`object_list`* | Shows only the paths to the named pins, ports, or endpoints clocked by named clocks. If you omit this option, all the paths that violate the timing constraints are shown. |
| `-no_environment` | Disables environment budgeting during checking. Use this option only when you use `check_budget` before budget allocation. When you use this option, you must use a cell list. If you omit this option, environment budgeting is performed. |
| `-no_interblock_logic` | Considers interblock logic during checking. Use this option only when you use `check_budget` before budget allocation. If you omit this option, interblock logic is considered fixed during checking. When you use this option, you must use a cell list. |
| *`cell_list`* | Lists the cells to budget. You must use a cell list if you use the `check_budget` command before you use the `dc_allocate_budgets` command. You must also use a cell list with the `-no_environment` and `-no_interblock_logic` options. |

Example 7-1 shows the `check_budget` command used with the `-verbose` and `-tolerance` options. Example 7-2 shows the `check_budget` command used with no options. See Appendix B, “set_user_budget, report_path_budget, and check_budget Examples” for additional examples.

*Example 7-1 check_budget -verbose -tolerance*

```
check_budget -verbose -tolerance 10 -from [all_inputs] -to [all_outputs]
****************************************
   Check budget
****************************************

  Startpoint: IN (input port)
  Endpoint: OUT (output port)
  Path Group: default
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  test              io30x30             cba_core

  Point                                 Incr       Path
  -----------------------------------------------------------
  input external delay                  0.00       0.00 f
  C1/IN (D1)                            0.00       0.00 f
  C1/OUT (D1)                           0.00       0.00 r
  C2/IN (D2)                            0.00       0.00 f
  C2/OUT (D2)                           0.00       0.00 r
  C3/IN (D3)                            0.00       0.00 f
  C3/OUT (D3)                           0.00       0.00 r
  data arrival time                                0.00

  max_delay                           100.00     100.00
  output external delay                 0.00     100.00
  data required time                             100.00
  -----------------------------------------------------------
  data required time                             100.00
  data arrival time                                0.00
  -----------------------------------------------------------
  slack (MET)                                    100.00

Design has 1 violations.
1
```

*Example 7-2 check_budget*

```
check_budget
****************************************
   Check budget
****************************************
Design has 1 violations.
1
```

## dc_allocate_budgets

The `dc_allocate_budgets` command automatically allocates budgets among specified cells or pins and to blocks contained within other blocks (hierarchical blocks). See Appendix A, "RTL Methodology Example," and Appendix B, "set_user_budget, report_path_budget, and check_budget Examples."

For hierarchical blocks, the budgeter takes a global view and allocates budgets recursively to specified blocks and the subblocks within them (down the hierarchy) to the limit you specify, then generates a constraint for each.

Note:

If you enable latch-based time borrowing before budgeting, `dc_allocate_budgets` uses the actual time borrowed to allocate budgets.

By default, `dc_allocate_budgets` deletes existing budget information and performs full budget allocation on the specified cells.

The syntax is

```
dc_allocate_budgets [-levels 1|2|...|all]
                  [-mode rtl|gate|mixed][-write_script]
[-format dcsh|dctcl][-file_format_spec format_spec]
[-separator name_separator][-no_interblock_logic]
[-budget_design_ware][-min_register_to_output
minimum_budget]
                  [-min_input_to_output minimum_budget]
[-min_input_to_register minimum_budget][cell_list]
```

| Argument | Description |
|---|---|
| `-levels 1\|2\|...\|all` | Enables the hierarchical budget allocation mode and specifies the number of levels of hierarchy for which to allocate budgets, starting from the current design. A positive integer specifies the number of levels down the hierarchy to which budgets are allocated. `-levels all` allocates budgets down all levels of hierarchy. Note that using `dc_allocate_budgets -levels all` is the same as `dc_allocate_budgets` because `dc_allocate_budgets` defaults to gate mode and allocates budgets to all cells in the design starting at the top level of hierarchy. See "Specifying the Levels of Logic to Budget" on page 3-2. |

| Argument | Description |
|---|---|
| `-mode rtl\|gate\|mixed` | Specifies the mode in which the budgeter runs.<br>The `rtl` mode budgets a completely or partially unmapped design. If the design contains mapped logic, the logic is treated as fixed and no slack is allocated to it.<br>The `gate` mode budgets a fully or mostly mapped design, allocating slack among mapped cells. No slack is allocated to unmapped (RTL) cells. This is the default mode.<br>The `mixed` mode budgets designs containing both RTL and mapped gate-level netlists. In general, mixed-mode typically improves QoR when your design contains over 30% RTL. The QoR improvements are due to the improved budgeting algorithms which might slow runtime up to 20%. See "Budgeting Mode" on page 2-2. |
| `-write_script` | Writes the context for each budgeted design to files. |
| `-format dctcl\|dcsh` | Specifies the output format for script files. The default is dctcl (for dc_shell-t and dc_shell-xg-t). |
| `-file_format_spec` *`format_spec`* | Specifies the format for the names of constraint files generated for each budgeted cell generated using the `-write_script` option. The default is ./%C.ptsh.<br>The format specification contains the directory to which the files are to be written, the conversion specification: %C (for cells) or %D (for designs), and the prefix, suffix and extension to use.<br>For the %D specification, the constraint file name is obtained by replacing the conversion specification by either the name of the cell design, if the design is referenced once (uniquified design), or by the concatenation of the design name and the cell's full name.<br>For the %C conversion specification (the default), the constraint file name is obtained by replacing the conversion specification by the cell's full name.<br>Example<br>Using `-file_format_spec dir1/fred_%D_pass1.scr` creates a constraint file named fred_design_name_pass1.scr in the directory dir1. The default value for format_spec, ./%C.ptsh, creates a constraint file named cell_name.ptsh in the current directory. If you specify a value for the -format option, the value you specify (dcsh or dctcl) is used as the extension. |

| Argument | Description |
|---|---|
| `-separator` *`name_separator`* | Specifies the single-character string to use as a hierarchical separator in constructing the derived constraint file name generated for each instance. The default is `@`. You cannot use the slash (/), percent (%), or backslash (\) characters as name separators. |
| `-no_interblock_logic` | Disables budgeting interblock (glue) logic. If you omit this option, the budgeter considers the interblock logic as variable and budgets interblock logic. See “Controlling the Budgeting of Interblock Logic” on page 3-3. |
| `-budget_design_ware` | Budgets DesignWare components. If you omit this option, these components are considered fixed and are not budgeted. Use this switch only for mapped designs. |
| `-min_register_to_ output minimum budget` | Specifies the minimum delay that can be assigned to a path between a register output and an output port.<br>Set the value of `-min_register_to_output` to the Clk-to-Q delay of a typical register. This prevents the budgeter from allocating a constraint of 0 to paths that directly connect register outputs to output ports. See “Setting Minimum Budgets on All Timing Paths” on page 3-12. Registering outputs is the Synopsys desired methodology. If the user specifies a min budget constraint that is larger than what the budgeter would have otherwise calculated, the min budget constraint typically takes precedence. If the user specifies a min budget that is smaller than what the budgeter would have done, then the user's min budget is ignored. Basically, the user's min budget constraint is treated as a lower bound on the delay of that path segment. |
| `-min_input_to_ register minimum budget` | Specifies a minimum delay that can be assigned to a path between input ports and register synchronous input pins.<br>Specify this delay to inputs and register synchronous inputs or to take account of simple gates between registers and inputs (gates such as inverters or buffers used to manage upstream loading). |

| Argument | Description |
|---|---|
| `-min_input_to_output minimum budget` | Specifies a minimum delay that can be assigned to combinational paths from input ports to output ports.<br>Specify this delay to provide for wire load delays on wires driving directly between ports or buffers that are used to prevent feed-through wires from being used. |
| *`cell_list`* | Lists the hierarchical cells to budget. The default is to allocate budgets to all cells in the design starting at the top level of hierarchy. |

## remove_user_budget

The `remove_user_budget` command removes user-specified budget information previously set using the `set_user_budget` command. The syntax is

```
remove_user_budget -from object_list -to object_list
```

| Argument | Description |
|---|---|
| `-from` *`object_list`* | Lists the pins or clocks from which to remove a user-specified budget. |
| `-to` *`object_list`* | Lists the pins or clocks to which you want to remove a user-specified budget. |

## report_path_budget

The `report_path_budget` command provides a report of budget distribution, delays, budget slack, and constraints for budgeted paths. Example 7-3 on page 7-11 shows budget report. See

Appendix A, "RTL Methodology Example," and Appendix B, "set_user_budget, report_path_budget, and check_budget Examples."

Slack is defined as the difference between the budgets and the delay of a path.

The report includes most information reported by `report_timing` plus the following information:

- Budget for each segment
- Budget slack for each path segment (difference between the budget assigned for a path segment and the delay of the path segment)
- Weight assigned to each path segment
- Attributes indicating whether the budgets allocated to each path segment are automatic budgets, user-specified budgets, or considered as fixed delay
- Total budget slack for an entire path (difference between budget arrival time and actual data arrival time for the budgeted netlist)

If you omit `-from`, `-to`, or `-through`, by default the path with the worst slack is reported.

```
report_path_budget[-from object_list] [-to object_list]
                  [-through object_list] [-nworst integer]
                  [-max_paths integer] [-nosplit]
                  [-verbose] [-all] [-sort_by group|slack]
```

| Argument | Description |
|---|---|
| `-from` *`object_list`* | Reports only paths from the listed pins, ports, or startpoints clocked by named clocks. If you do not specify an object list, the default behavior reports the path with the worst slack. |
| `-to` *`object_list`* | Reports only paths to the listed pins, ports, or endpoints clocked by named clocks. If you do not specify an object list, the default behavior reports the path with the worst slack. |
| `-through` *`object_list`* | Reports only objects through the listed pins or ports. If you do not specify an object list, the default behavior reports the path with the worst slack. |
| `-max_paths` *`integer`* | Specifies the number of paths to report. The default is 1. The range is 1 to 2,000,000. |
| `-nworst` *`integer`* | Specifies the number of paths to report per endpoint. The default is 1. The range is 1 to 2,000,000. |
| `-nosplit` | Disables line splitting when information exceeds the column's width. |
| `-verbose` | Lists leaf-level cells in the path and their budgets. |
| `-all` | Lists the budgets for all hierarchical cells in the path. If you omit this option, only cells that are budgeted are listed. |
| `-sort_by group\|slack` | Specifies the order in which the paths are reported. By default, paths are sorted by costing groups, and within each group the paths are ordered by slack. With the slack option, the paths are ordered by slack only. The default is `-sort_by group`. |

## *Example 7-3 report_path_budget*

```
report_path_budget -verbose

****************************************
Report : budget
```

```
        -path full
        -delay max
        -max_paths 1
Design : cpu_ctl
Version: V-2004.06
Date   : Tue Mar 23 11:52:08 2004
****************************************

Attributes
-----------
U   User Specified Budgets
A   Automatic Budgets
F   Fixed delay

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: nom_pvt   Library: dsp_c035_lc_lib
Wire Load Model Mode: enclosed

  Startpoint: ctl_pipe_ctl0/emu_dfc_reg0/u1_5
              (rising edge-triggered flip-flop clocked by clock)
  Endpoint: cpu_to_emu_edi_abort
            (output port clocked by virtual_clock)
  Path Group: virtual_clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  cpu_ctl            0.5K                  dsp_c035_lc_lib
  ctl_pipe_ctl_vec_reg_5_0
                     0.5K                  dsp_c035_lc_lib
  ctl_pipe_ctl       0.5K                  dsp_c035_lc_lib

  Point                  Budget     Delay     Slack  Type
  -------------------------------------------------------
  clock clock (rise edge)
                           0.00      0.00      0.00  F
  clock network delay      0.00      0.00      0.00  F
  ctl_pipe_ctl0/emu_dfc_reg0/u1_5/phase1_clock
                           0.00      0.00      0.00  A
  ctl_pipe_ctl0/emu_dfc_reg0/u1_5/q
                           9.04    110.27   -101.23  A
  ctl_pipe_ctl0/emu_dfc_reg0/U5/a
                           0.13      0.13      0.00  F
  ctl_pipe_ctl0/emu_dfc_reg0/U5/y
                           8.90     69.18    -60.27  A
  ctl_pipe_ctl0/emu_dfc_reg0/outvec[4]
                           0.00      0.00      0.00  F
  ctl_pipe_ctl0/U334/a0
                         2.84      2.84     -0.00  F
  ctl_pipe_ctl0/U334/y
                       227.07    269.02    -41.95  A
  ctl_pipe_ctl0/U544/a
                         0.10      0.10     -0.00  F
  ctl_pipe_ctl0/U544/y
                        35.87    118.36    -82.49  A
  ctl_pipe_ctl0/U543/b
                         1.22      1.22      0.00  F
  ctl_pipe_ctl0/U543/y
                         7.82     57.42    -49.61  A
  ctl_pipe_ctl0/U308/a
                         0.56      0.56     -0.00  F
  ctl_pipe_ctl0/U308/y
                       105.67     60.71     44.96  A
  ctl_pipe_ctl0/U307/a
                         0.20      0.20     -0.00  F
  ctl_pipe_ctl0/U307/y
                        76.67    102.53    -25.86  A
  ctl_pipe_ctl0/cpu_to_emu_edi_abort
                         0.00      0.00      0.00  F
  cpu_to_emu_edi_abort
                         1.99      1.99      0.00  F
  ------------------------------------------------------
  Total                478.07    794.52   -316.45
  Required time        550.00    550.00
  ------------------------------------------------------
  Slack                 71.93   -244.52


1

1
```

## set_context_margin

The `set_context_margin` command specifies a margin to add to or subtract from input and output delay values generated by the `dc_allocate_budgets -write_script` command.

Specify the margin as an absolute value or as a percentage of the constraint.

By default, input and output delays are adjusted to be more constraining (the specified margin is added to the maximum delay values and subtracted from the minimum delay values). To cause the margin to be subtracted from the maximum delay values and added to the minimum delay values, use the `-relax` option.

When you use the `set_context_margin` command, keep the following points in mind:

- If you omit the object list, the margin applies to the current design.
- If you specify a pin but do not specify a value, the value set for the parent cell is used.
- If neither the pin nor the parent cell have a value set, the value set for the current design is used.
- If no margin is specified, the tool uses the default value of 0.0.

The syntax is

```
set_context_margin [-percent] [-relax] [-max] value
                       [object_list]
```

| **Argument** | **Description** |
|---|---|
| `-percent` | Specifies that the value is a percentage of the delay. |
| `-relax` | Relaxes (rather than tightens) the constraint. |
| `-max` | Specifies the margin for maximum constraints. |
| *value* | Sets the budget value or percentage. |
| *object_list* | Lists the cells or pins. If you omit this option, the margin applies to the current design. |

## set_user_budget

The `set_user_budget` command specifies user directives for budget allocation. The syntax is

```
set_user_budget -from object_list -to object_list
                  -percent value
```

| Argument | Description |
|---|---|
| `-from` *`object_list`* | Lists the pins or clocks from which to budget. |
| `-to` *`object_list`* | Lists the pins or clocks to which to budget. |
| `-percent` | Specifies the budget value as a percentage. |
| *`value`* | Sets the value. |

For examples, see "Setting a User Budget on a Timing Path" on page 3-10 and Appendix B, "set_user_budget, report_path_budget, and check_budget Examples."

## budget_generate_critical_range

When set to true, the `budget_generate_critical_range` variable enables the budgeter to propagate 10% of the shortest clock period in the design to multiply instantiated budgeted blocks. The default value is false. If `set_critical_range` is specified in the top-level constraint file, this value overrides the budgeter generated `set_critical_range` constraints and is propagated to all the subdesigns. For variable details, see the man pages.

# A

# RTL Methodology Example

This appendix describes an RTL budgeting design example in the following sections:

- RTL Designs: top, frontpipe, middlepipe, and rearpipe
- Top-Level Constraints
- Budgeting Session Log
- Constraint Files Created by RTL Budgeting
- Reviewing Path Budgets
- Compiling the Design With Budgeted Constraint Files

This example follows the methodology described in “RTL Budgeting Mode Methodology” on page 2-2.

## RTL Designs: top, frontpipe, middlepipe, and rearpipe

The top-level design, top, contains three designs:

- frontpipe
- middlepipe
- rearpipe

The RTL code for the designs is shown in Examples A-1 through A-4.

*Example A-1 Design: top RTL*

```
module top (in_1, clk, out);

   input in_1;
   input   clk;
   output out;
   wire t1,t2,t3,t4;

assign t2 = ~ t1;
assign t4 = ~ t3;

   frontpipe u1 (
                  .in_f(in_1),
                  .out_f (t1),
                  .clk(clk)
                 );

   middlepipe u2 (
                  .in_m(t2),
                  .out_m(t3),
                  .clk(clk)
                 );

   rearpipe u3  (
                  .in_r(t4),
                  .clk(clk),
                  .out_r(out)
                 );
endmodule
```

*Example A-2 Design: frontpipe RTL*

```
module frontpipe (in_f, clk, out_f);
   input  in_f;
   input clk;
   output   out_f;

   reg  ff_f;
   wire f1, f2, f3;


   always @ (posedge clk)
   begin
       ff_f <= in_f;
      end

assign out_f = ~ ff_f ;

endmodule
```

*Example A-3 Design: middlepipe RTL*

```
module middlepipe ( in_m, clk, out_m);
   input  clk, in_m;
   output   out_m;
   reg ff_m;

   always @ (posedge clk)
   begin
       ff_m <= in_m;
      end

  assign out_m = ~ ff_m;
endmodule
```

*Example A-4 Design: rearpipe RTL*

```
module rearpipe (in_r, clk, out_r);
  input in_r;
  input clk;
  output  out_r;

  reg  ff_r;
  wire out_r,r1;
  assign r1 =  ~ in_r;

  always @ (posedge clk)
  begin
       ff_r <= r1;
  end

  assign out_r = ~ff_r;

endmodule
```

## Top-Level Constraints

The top-level constraints for design top are shown in Example A-5.

*Example A-5  Top-Level Constraints*

```
set CLK_PERIOD 10.0
set IN_DELAY 1.5
set OUT_DELAY 2
create_clock -period $CLK_PERIOD -name clk [get_ports clk]
set_dont_touch_network [get_clocks clk]
set_input_delay $IN_DELAY -max -clock clk [remove_from_collection [all_inputs]
[get_ports clk]]
set_output_delay $OUT_DELAY -max -clock clk [all_outputs]
set_operating_conditions typ_0_1.98
set_wire_load_model -name  10KGATES
```

## Budgeting Session Log

Example A-6 shows `dc_allocate_budgets` usage.

*Example A-6  Budgeting Log*

```
read_verilog a.v
Loading verilog file '/...../a.v'
Detecting input file type automatically (-rtl or -netlist).
Reading with Presto HDL Compiler (equivalent to -rtl option).
Running PRESTO HDLC
Loading db file '/...... syn/standard.sldb'
Loading db file '/..... gtech.db'
Loading db file '/...../core_typ.db'
Compiling source file /...../a.v

Inferred memory devices in process
        in routine frontpipe line 31 in file
             '/...../a.v'.
==============================================================================
|    Register Name    |    Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
==============================================================================
|      ff_f_reg       | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
==============================================================================

Inferred memory devices in process
        in routine middlepipe line 42 in file
             '/.../a.v'.
==============================================================================
|    Register Name    |    Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
==============================================================================
|      ff_m_reg       | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
==============================================================================

Inferred memory devices in process
        in routine rearpipe line 56 in file
             '/.../a.v'.
==============================================================================
|    Register Name    |    Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
==============================================================================
|      ff_r_reg       | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
==============================================================================
Presto compilation completed successfully.
Current design is now '/.../top.db:top'
Loaded 4 designs.
top frontpipe middlepipe rearpipe
link

  Linking design 'top'
  Using the following designs and libraries:
  --------------------------------------------------------------------------
  * (4 designs)                   /.../top.db, etc.
  ssc_core (library)              /.../libs/core_typ.db

1
source top.tcl
Using operating conditions 'typ_0_1.98' found in library 'ssc_core'.
Design top: Using wire_load model '10KGATES' found in library 'ssc_core'.
1
dc_allocate_budgets -level 1 -mode rtl -write_script -no_interblock_logic
-file_format_spec %D.rtl.con
```

```
  Loading design 'top'
Information: Changed wire load model for 'rearpipe' from '(none)' to '5KGATES'.
(OPT-170)
Information: Changed wire load model for 'middlepipe' from '(none)' to '5KGATES'.
(OPT-170)
Information: Changed wire load model for 'frontpipe' from '(none)' to '5KGATES'.
(OPT-170)
  Generating constraints using cell 'u1' for design 'frontpipe'
  Generating constraints using cell 'u2' for design 'middlepipe'
  Generating constraints using cell 'u3' for design 'rearpipe'
1
```

## Constraint Files Created by RTL Budgeting

The constraint files generated by the RTL budgeting in Example A-6 are shown in Example A-7, Example A-8, and Example A-9.

*Example A-7 frontpipe Constraints*

```
#####################################################

# Created by write_script() -format dctcl on Tue Mar 23 11:34:54 2004

#####################################################

# Set the current_design #
current_design frontpipe

create_clock -period 10 -waveform {0 5} [get_ports {clk}]
set_dont_touch_network [get_clocks {clk}]
set_input_delay 1.5 -clock "clk" [get_ports {in_f}]
set_output_delay 9.02758 -clock "clk" [get_ports {out_f}]
set_operating_conditions "typ_0_1.98" -library "ssc_core"
set_wire_load_mode "enclosed"
set_wire_load_selection_group "AreaBasedWireLoadSelection"
```

*Example A-8 middlepipe Constraints*

```
#####################################################

# Created by write_script() -format dctcl on Tue Mar 23 11:34:54 2004

#####################################################
```

```
# Set the current_design #
current_design middlepipe

create_clock -period 10 -waveform {0 5} [get_ports {clk}]
set_dont_touch_network [get_clocks {clk}]
set_input_delay 0.972424 -clock "clk" [get_ports {in_m}]
set_output_delay 9.5 -clock "clk" [get_ports {out_m}]
set_operating_conditions "typ_0_1.98" -library "ssc_core"
set_wire_load_mode "enclosed"
set_wire_load_selection_group "AreaBasedWireLoadSelection"
set_driving_cell -lib_cell inv1a3 [get_ports {in_m}]
```

*Example A-9 rearpipe Constraints*

```
 #####################################################

# Created by write_script() -format dctcl on Tue Mar 23 11:34:54 2004

#####################################################

# Set the current_design #
current_design rearpipe

create_clock -period 10 -waveform {0 5} [get_ports {clk}]
set_dont_touch_network [get_clocks {clk}]
set_input_delay 0.393312 -clock "clk" [get_ports {in_r}]
set_output_delay 2 -clock "clk" [get_ports {out_r}]
set wire_load_mode "enclosed"
set_wire_load_selection_group "AreaBasedWireLoadSelection"
set_driving_cell -lib_cell inv1a3 [get_ports {in_r}]
```

## Reviewing Path Budgets

After you allocate budgets, use the `report_path_budget` command to check timing as shown in Example A-10.

*Example A-10 report_path_budget*

```
report_path_budget -max_path 4 -all
```

```
****************************************
Report : budget
        -path full
        -delay max
        -max_paths 4
Design : top
Version: V-2004.06-LS-BETA2
Date   : Tue Mar 23 11:34:55 2004
****************************************

Attributes
-----------
U   User Specified Budgets
A   Automatic Budgets
F   Fixed delay

Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed

  Startpoint: u3/ff_r_reg
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: out (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Point                 Budget     Delay     Slack  Type
  ------------------------------------------------------
  clock clk (rise edge)
                          0.00      0.00      0.00  F
  clock network delay     0.00      0.00      0.00  F
  u3/ff_r_reg/clocked_on
                          0.00      0.00      0.00  A
  u3/ff_r_reg/Q           4.67      0.00      4.67  A
  u3/out_r                3.33      0.00      3.33  A
  ------------------------------------------------------
  Total                   8.00      0.00      8.00
  Required time           8.00      8.00
  ------------------------------------------------------
  Slack                   0.00      8.00


  Startpoint: in_1 (input port clocked by clk)
  Endpoint: u1/ff_f_reg
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Point                 Budget     Delay     Slack  Type
```

```
------------------------------------------------------
clock clk (rise edge)
                         0.00      0.00      0.00  F
clock network delay      0.00      0.00      0.00  F
input external delay
                         1.50      1.50      0.00  F
u1/in_f                  0.00      0.00      0.00  F
u1/ff_f_reg/next_state
                         0.00      0.00      0.00  F
------------------------------------------------------
Total                    1.50      1.50      0.00
Required time           10.00     10.00
------------------------------------------------------
Slack                    8.50      8.50


Startpoint: u1/ff_f_reg
            (rising edge-triggered flip-flop clocked by clk)
Endpoint: u2/ff_m_reg
          (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Point                  Budget     Delay     Slack  Type
------------------------------------------------------
clock clk (rise edge)
                         0.00      0.00      0.00  F
clock network delay      0.00      0.00      0.00  F
u1/ff_f_reg/clocked_on
                         0.00      0.00      0.00  A
u1/ff_f_reg/Q            0.56      0.00      0.56  A
u1/out_f                 0.42      0.00      0.42  A
u2/in_m                  0.00      0.00      0.00  F
u2/ff_m_reg/next_state
                         0.00      0.00      0.00  F
------------------------------------------------------
Total                    0.97      0.00      0.97
Required time           10.00     10.00
------------------------------------------------------
Slack                    9.03     10.00


Startpoint: u2/ff_m_reg
            (rising edge-triggered flip-flop clocked by clk)
Endpoint: u3/ff_r_reg
          (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max
```

```
  Point                  Budget     Delay     Slack  Type
  ------------------------------------------------------
  clock clk (rise edge)
                           0.00      0.00      0.00  F
  clock network delay      0.00      0.00      0.00  F
  u2/ff_m_reg/clocked_on
                           0.00      0.00      0.00  A
  u2/ff_m_reg/Q            0.22      0.00      0.22  A
  u2/out_m                 0.18      0.00      0.18  A
  u3/in_r                  0.00      0.00      0.00  F
  u3/ff_r_reg/next_state
                           9.32      0.00      9.32  A
  ------------------------------------------------------
  Total                    9.71      0.00      9.71
  Required time           10.00     10.00
  ------------------------------------------------------
  Slack                    0.29     10.00


1
```

## Compiling the Design With Budgeted Constraint Files

After reviewing the path budgets produced by the `report_path_budget` command, compile the designs as shown in Example A-11. Note that

```
 dc_allocate_budgets -level 1 -mode rtl -write_script
-no_interblock_logic -file_format_spec %D.rtl.con
```

creates constraint file names of frontpipe.rtl.con, middlepipe.rtl.con, and rearpipe.rtl.con.

*Example A-11    Compile Designs: frontpipe, middlepipe, rearpipe, and top*

```
current_design rearpipe
Current design is 'rearpipe'.
{"rearpipe"}
source rearpipe.rtl.con
Current design is 'rearpipe'.
```

```
Using operating conditions 'typ_0_1.98' found in library 'ssc_core'.
Warning: Design rule attributes from the driving cell will be
        set on the port. (UID-401)
1
compile -map medium
Information: Evaluating DesignWare library utilization. (UISN-27)

============================================================================
| DesignWare Building Block Library       |      Version       | Available |
============================================================================
| Basic DW Building Blocks                |
                                         V-2004.06-DWF_0406    |     *     |
| Licensed DW Building Blocks             |
                                                               |           |
============================================================================

  Loading target library 'ssc_core'

  Beginning Pass 1 Mapping
  ------------------------
  Processing 'rearpipe'
.....
.....
  Optimization Complete
  ---------------------
  Transferring design 'rearpipe' to database 'rearpipe.db'

Current design is 'rearpipe'.
1
set_dont_touch rearpipe
1
current_design middlepipe
Current design is 'middlepipe'.
{"middlepipe"}
source middlepipe.rtl.con
Current design is 'middlepipe'.
Using operating conditions 'typ_0_1.98' found in library 'ssc_core'.
Warning: Design rule attributes from the driving cell will be
        set on the port. (UID-401)
1
compile -map medium

  Loading target library 'ssc_core'

  Beginning Pass 1 Mapping
  ------------------------
  Processing 'middlepipe'
...
```

```
...
  Optimization Complete
  ---------------------
  Transferring design 'middlepipe' to database 'middlepipe.db'

Current design is 'middlepipe'.
1
set_dont_touch middlepipe
1
current_design  frontpipe
Current design is 'frontpipe'.
{"frontpipe"}
source frontpipe.rtl.con
Current design is 'frontpipe'.
Using operating conditions 'typ_0_1.98' found in library 'ssc_core'.
1
compile -map medium

  Loading target library 'ssc_core'

  Beginning Pass 1 Mapping
  ------------------------
  Processing 'frontpipe'
...
...
  Optimization Complete
  ---------------------
  Transferring design 'frontpipe' to database 'frontpipe.db'

Current design is 'frontpipe'.
1
set_dont_touch frontpipe
1
current_design top
Current design is 'top'.
{"top"}
compile -map medium

...
...
  Optimization Complete
  ---------------------
  Transferring design 'top' to database 'top.db'

Current design is 'top'.
1
current_design rearpipe
Current design is 'rearpipe'.
```

```
{"rearpipe"}
set_dont_touch rearpipe false
1
current_design middlepipe
Current design is 'middlepipe'.
{"middlepipe"}
set_dont_touch middlepipe false
1
current_design frontpipe
Current design is 'frontpipe'.
{"frontpipe"}
set_dont_touch frontpipe false
1
current_design top
Current design is 'top'.
{"top"}
compile -map medium -top

...
...
  Optimization Complete
  ---------------------
  Transferring design 'top' to database 'top.db'

Current design is 'top'.
1
write_file -f db -hier -o gate.db
Writing to file /.../gate.db
1
source top.tcl
Using operating conditions 'typ_0_1.98' found in library 'ssc_core'.
Design top: Using wire_load model '10KGATES' found in library 'ssc_core'.
1
```

# B

# set_user_budget, report_path_budget, and check_budget Examples

This appendix shows the `set_user_budget` command usage, the `report_path_budget` and `check_budget` outputs, and the constraint files generated by gate-level budgeting.

The design used for the Appendix B gate-level budgeting example is the compiled design from Appendix A. Example A-5 shows the top-level constraints.

This appendix includes the following sections:

## set_user_budget Example

To set user specified budgets, use the `set_user_budget` command as shown in Example B-1.

*Example B-1 set_user_budget*

```
set_user_budget -from u2/clk -to u2/* 20
1
dc_allocate_budgets -level 1 -mode gate -write_script
  Loading design 'top'
  Characterizing cell 'u1' on design 'frontpipe'
  Characterizing cell 'u2' on design 'middlepipe'
  Characterizing cell 'u3' on design 'rearpipe'
```

## report_path_budget Example

To identify user budgets, a U is inserted in the Type column in the `report_path_budget` report as shown in Example B-2. In this design, the user budget caused negative slack on a path.

*Example B-2 report_path_budget: Contains User Budget*

```
report_path_budget -max 4 -verbose

****************************************
Report : budget
        -path full
        -delay max
        -max_paths 4
Design : top
Version: V-2004.06
Date   : Tue Mar 23 11:35:11 2004
****************************************

Attributes
-----------
U   User Specified Budgets
A   Automatic Budgets
```

```
F   Fixed delay

Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed

  Startpoint: u3/ff_r_reg
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: out (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  top                10KGATES              ssc_core
  rearpipe           5KGATES               ssc_core

  Point                 Budget     Delay     Slack  Type
  ------------------------------------------------------
  clock clk (rise edge)
                          0.00      0.00      0.00  F
  clock network delay     0.00      0.00      0.00  F
  u3/ff_r_reg/CLK         0.00      0.00      0.00  A
  u3/ff_r_reg/Q           5.49      0.48      5.01  A
  u3/U5/A                 0.00      0.00     -0.00  F
  u3/U5/Y                 2.51      0.10      2.40  A
  u3/out_r                0.00      0.00      0.00  F
  out                     0.00      0.00      0.00  F
  ------------------------------------------------------
  Total                   8.00      0.59      7.41
  Required time           8.00      8.00
  ------------------------------------------------------
  Slack                   0.00      7.41


  Startpoint: in_1 (input port clocked by clk)
  Endpoint: u1/ff_f_reg
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  top                10KGATES              ssc_core

  Point                 Budget     Delay     Slack  Type
  ------------------------------------------------------
  clock clk (rise edge)
                          0.00      0.00      0.00  F
```

```
clock network delay     0.00      0.00      0.00  F
input external delay
                        1.50      1.50      0.00  F
in_1                    0.00      0.00      0.00  F
u1/in_f                 0.00      0.00      0.00  F
u1/ff_f_reg/D           0.00      0.00      0.00  F
-------------------------------------------------------
Total                   1.50      1.50      0.00
Required time           9.88      9.88
-------------------------------------------------------
Slack                   8.38      8.38


Startpoint: u1/ff_f_reg
            (rising edge-triggered flip-flop clocked by clk)
Endpoint: u2/ff_m_reg
          (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Des/Clust/Port     Wire Load Model       Library
-------------------------------------------------
top                10KGATES              ssc_core
frontpipe          5KGATES               ssc_core

Point                  Budget     Delay     Slack  Type
-------------------------------------------------------
clock clk (rise edge)
                        0.00      0.00      0.00  F
clock network delay     0.00      0.00      0.00  F
u1/ff_f_reg/CLK         0.00      0.00      0.00  A
u1/ff_f_reg/Q           5.04      0.48      4.56  A
u1/U4/A                 0.00      0.00     -0.00  F
u1/U4/Y                 2.43      0.12      2.31  A
u1/out_f                0.00      0.00      0.00  F
U4/A                    0.00      0.00      0.00  F
U4/Y                    2.40      0.12      2.29  A
u2/in_m                 0.00      0.00      0.00  F
u2/ff_m_reg/D           0.00      0.00      0.00  F
-------------------------------------------------------
Total                   9.87      0.72      9.16
Required time           9.87      9.87
-------------------------------------------------------
Slack                   0.00      9.16


Startpoint: u2/ff_m_reg
            (rising edge-triggered flip-flop clocked by clk)
```

```
  Endpoint: u3/ff_r_reg
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  top                10KGATES              ssc_core
  middlepipe         5KGATES               ssc_core
  rearpipe           5KGATES               ssc_core

  Point                  Budget      Delay      Slack  Type
  ------------------------------------------------------
  clock clk (rise edge)
                           0.00       0.00       0.00  F
  clock network delay      0.00       0.00       0.00  F
  u2/ff_m_reg/CLK          0.00       0.00       0.00  A
  u2/ff_m_reg/Q           10.00       0.37       9.63  U
  u2/U4/A                  0.00       0.00      -0.00  F
  u2/U4/Y                 10.00       0.10       9.90  U
  u2/out_m                 0.00       0.00       0.00  F
  U3/A                     0.00       0.00       0.00  F
  U3/Y                     0.01       0.11      -0.10  A
  u3/in_r                  0.00       0.00       0.00  F
  u3/U6/A                  0.00       0.00       0.00  F
  u3/U6/Y                  0.01       0.11      -0.10  A
  u3/ff_r_reg/D            0.00       0.00      -0.00  F
  ------------------------------------------------------
  Total                   20.02       0.71      19.32
  Required time            9.87       9.87
  ------------------------------------------------------
  Slack                  -10.15       9.17
```

## check_budget Example

The user budget caused a design violation—identified by the `check_budget` command shown in Example B-3.

*Example B-3 check_budget: Design Has One Violation*

```
check_budget -verbose
****************************************
   Check budget
****************************************
```

```
  Startpoint: u2/ff_m_reg
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: u3/ff_r_reg
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  top                10KGATES              ssc_core
  middlepipe         5KGATES               ssc_core
  rearpipe           5KGATES               ssc_core

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  u2/ff_m_reg/CLK (fdf1a6)                 0.00       0.00 r
  u2/ff_m_reg/Q (fdf1a6)                  10.00 *    10.00 r
  u2/out_m (middlepipe)                   10.00      20.00 f
  u3/in_r (rearpipe)                       0.01      20.01 r
  u3/ff_r_reg/D (fdf1a3)                   0.01      20.02 f
  data arrival time                                  20.02

  clock clk (rise edge)                   10.00      10.00
  clock network delay (ideal)              0.00      10.00
  u3/ff_r_reg/CLK (fdf1a3)                 0.00      10.00 r
  library setup time                      -0.13       9.87
  data required time                                  9.87
  -----------------------------------------------------------
  data required time                                  9.87
  data arrival time                                 -20.02
  -----------------------------------------------------------
  slack (VIOLATED)                                  -10.15

Design has 1 violations.
```

## Constraint Files From Gate-Level Budgeting

The constraint files generated from the gate-level budgeting in Example B-1 are shown in examples B-4 through B-6.

## *Example B-4 frontpipe Constraint File*

```
#####################################################

# Created by write_script() -format dctcl on Tue Mar 23 11:35:11 2004

#####################################################

# Set the current_design #
current_design frontpipe

create_clock -period 10 -waveform {0 5} [get_ports {clk}]
set_dont_touch_network [get_clocks {clk}]
set_input_delay 1.5 -clock "clk" [get_ports {in_f}]
set_output_delay 2.53092 -rise -clock "clk" [get_ports {out_f}]
set_output_delay 2.49063 -fall -clock "clk" [get_ports {out_f}]
set_local_link_library {core_typ.db}
set_operating_conditions "typ_0_1.98" -library "ssc_core"
set_wire_load_mode "enclosed"
set_wire_load_selection_group "AreaBasedWireLoadSelection"
set_wire_load_selection_group -min "AreaBasedWireLoadSelection"
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {in_f}]
set_connection_class "universal" [get_ports {in_f}]
set_load -pin_load 0.006 [get_ports {clk}]
set_port_fanout_number 2 [get_ports {clk}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {clk}]
set_connection_class "default" [get_ports {clk}]
set_load -pin_load 0.007 [get_ports {out_f}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {out_f}]
set_fanout_load 0 [get_ports {out_f}]
set_connection_class "default" [get_ports {out_f}]
```

## *Example B-5 middlepipe Constraint File*

```
#####################################################

# Created by write_script() -format dctcl on Tue Mar 23 11:35:11 2004

#####################################################

# Set the current_design #
current_design middlepipe

create_clock -period 10 -waveform {0 5} [get_ports {clk}]
set_dont_touch_network [get_clocks {clk}]
set_input_delay 9.5 -clock "clk" [get_ports {in_m}]
set_output_delay 0 -clock "clk" [get_ports {out_m}]
```

```
set_local_link_library {core_typ.db}
set_operating_conditions "typ_0_1.98" -library "ssc_core"
set_wire_load_mode "enclosed"
set_wire_load_selection_group "AreaBasedWireLoadSelection"
set_wire_load_selection_group -min "AreaBasedWireLoadSelection"
set_driving_cell -lib_cell inv1a3 -library ssc_core -pin Y -from_pin A \
-input_transition_rise 0.0439622 -input_transition_fall 0.0294808 \
-no_design_rule [get_ports {in_m}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {in_m}]
set_max_capacitance 0.409 [get_ports {in_m}]
set_connection_class "default" [get_ports {in_m}]
set_load -pin_load 0.006 [get_ports {clk}]
set_port_fanout_number 2 [get_ports {clk}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {clk}]
set_connection_class "default" [get_ports {clk}]
set_load -pin_load 0.007 [get_ports {out_m}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {out_m}]
set_fanout_load 0 [get_ports {out_m}]
set_connection_class "default" [get_ports {out_m}]
```

## *Example B-6 rearpipe Constraint File*

```
#####################################################

# Created by write_script() -format dctcl on Tue Mar 23 11:35:11 2004

#####################################################

# Set the current_design #
current_design rearpipe

create_clock -period 10 -waveform {0 5} [get_ports {clk}]
set_dont_touch_network [get_clocks {clk}]
set_input_delay 9.5 -clock "clk" [get_ports {in_r}]
set_output_delay 2 -clock "clk" [get_ports {out_r}]
set_local_link_library {core_typ.db}
set_operating_conditions "typ_0_1.98" -library "ssc_core"
set_wire_load_mode "enclosed"
set_wire_load_selection_group "AreaBasedWireLoadSelection"
set_wire_load_selection_group -min "AreaBasedWireLoadSelection"
set_driving_cell -lib_cell inv1a3 -library ssc_core -pin Y -from_pin A \
-input_transition_rise 0.0424539 -input_transition_fall 0.0270482 \
-no_design_rule [get_ports {in_r}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {in_r}]
set_max_capacitance 0.409 [get_ports {in_r}]
set_connection_class "default" [get_ports {in_r}]
set_load -pin_load 0.006 [get_ports {clk}]
set_port_fanout_number 2 [get_ports {clk}]
```

```
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {clk}]
set_connection_class "default" [get_ports {clk}]
set_wire_load_model -name "10KGATES" -library "ssc_core"  [get_ports {out_r}]
set_fanout_load 0 [get_ports {out_r}]
set_connection_class "universal" [get_ports {out_r}]
```

# Index